Learnability and Characterization Results for Classes of Boolean Functions

Balázs Szörényi
Research Group on Artificial Intelligence

Advisor: György Turán

November 2007

A DISSERTATION SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY OF THE UNIVERSITY OF SZEGED

University of Szeged
Doctoral School in Mathematics and Computer Science
Ph.D. Program in Informatics

Preface

Revision can be thought of as the update of some existing (but somewhat erroneus) rule system, like some expert system provided by an expert. This problem arises when the rule system used to be correct, but the circumstances have changed, or when the rule system was erroneus initially. The present dissertation discusses this topic from the theoretical point of view, examining the possibility of efficient revision of some rule systems based on Boolean formulas, such as read-once formulas, projective DNF and threshold functions.

Additionally, characterization results are provided for some Boolean functions. Motivated by one of the revision algorithms, a structural description of a class of projective DNF is given. We also consider k-term DNF, and give a complete description of those formulas which have the largest number of prime implicants. This completes a series of well-known results on this class. A related characterization result is given for a class of DNF tautologies with a distance condition. Finally, motivated by a problem in belief revision (an area related to, but distinct from, theory revision), a criterion is given for the existence of a complement of a Horn formula.

Acknowledgement

First of all I would like to thank my advisor György Turán for helping and supporting me and conducting my research; I feel especially fortunate for having him as an advisor.

I would also like to thank Robert H. Sloan, with whom the work was most inspiring, and who helped me a lot during the common work and my travels to Chicago.

I am most thankful to János Csirik for letting me work at the Research Group on Artificial Intelligence, making it possible for me to concentrate on my scientific research.

I am also grateful to Boglárka Tóth, Zsolt Gazdag and Tamás Vinkó, with whom I shared the same office during the Ph.D. years; their friendship is invaluable.

Finally, my deepest thank goes to my whole family for their unconditional love, constant support and infinite belief in me.

Contents

Chapter 1

Introduction

The present dissertation, in its first part, considers theoretical results from the field of theory revision. Theory revision, as part of learning theory, is interested in reconstructing some unknown function acquiring information about it via some protocol, specified by the given learning model. However, as opposed to the general learning problem, it is assumed that the learner is not new to the given task, but it initially has a hypotheses that is assumed to be some rough approximation of the unknown function. As an analogous real-world example, one can consider an initial version of an expert system provided by an expert, which needs to be refined using further examples or other information available. Having some initial hypotheses available should make the learning problem easier to solve—making the relevance of the model apparent, and motivating its analysis from the theoretical point of view.

The theory revision results in the present dissertation all consider some Boolean formula class; read-once, threshold and projective DNF formulas [1] are analyzed from the point of view of efficient revisability.

In the second part characterizational results are presented; all showing equivalence between some syntactical and some semantical properties of some classes of Boolean functions. The syntactic properties involve Boolean formula classes, like DNFs satisfying some syntactic irredundancy notion, Horn formulas (one of the most studied formula class in artificial intelligence), disjoint DNFs (DNFs with pairwise conflicting terms) and decision trees (another very important object in computer science—which can also be thought of as a subclass of DNFs). The semantic properties include restrictions given for partitioning the n-dimensional cube with subcubes, special local restrictions given for a Boolean function on its domain, extensions of the truth set of some function fulfilling some special criteria, and finally some extremal properties.

[1]The class of projective DNF formulas form a new subclass of DNF formulas introduced recently by Valiant [128].

1.1 Learning and Theory Revision

Theory revision, or more generally, the whole area of learning theory aims to capture real life learning: to build models for some phenomena by collecting data about it and trying to generalize from this data by realizing regularities and extracting certain rules. An obvious and noble motivation for this is to make computers able to learn: to adapt to new situations in a changing environment. This as is one of the most fundamental original objectives of artificial intelligence. However, a big majority of real-world applications nowadays consist of problems that seem a bit different at first glance: to put up rules for, and to model systems that are way too complex for humans to do it by hand. Typical examples from everyday life are speech recognition, face recognition; or applications from bioinformatics like protein classification—and so on. Many of these tasks can be considered as the problem of finding a classification rule on a given domain that fits the data (i.e., labeling bitmaps either "woman" or "man", or mapping segments of speech to words, etc).

Various definitions and approaches were born to formulate this problem more precisely, but without a real consensus. However let us quote one (from Mitchell [99]):

> "A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Although it gives some intuition about the nature of learning as a mathematical problem, apparently it is too general to be applicable for specific problems or situations; so more formal definitions are needed.

Computational learning theory and its central notion, PAC learnability (established by the seminal paper [127] of Valiant), approaches learning from complexity theoretic point of view and is interested in the computational and information theoretic aspects of learning: what can be learned efficiently, and how much information does the learner need for this in different settings. (In this case the classification rules are often Boolean formulas from some predefined class.) Computational learning theory is defined in some sense as the inverse of cryptography—and indeed, subsequently Kearns and Valiant has shown that an efficient PAC learning algorithm for general Boolean formulas could be used for example to break RSA [77].

To mention some other fields also devoted to learning: in the framework of "learning in the limit" (established by Gold [47]) the learner meets in the course of an infinite process all the words (or expressions) of some language [2], and is required to set up a hypotheses: some representation of the language. On the other hand, pattern recognition, for example (highly influenced by works like that of Vapnik and Chervonenkis [131] and Stone [120]), is interested in classifiers that (constructing their hypotheses using randomly generated examples often in a kind of on-line manner) are asymptotically as good as the best possible (called **Bayes classifier**).

[2]And, depending on the specific model, the learner might additionally meet some or all of the "negative" examples: words or expressions not in the given language.

The research aimed to analyze different aspects of PAC learnability gave birth to several other related learning models. On the whole—focusing on the Boolean case—all of them are interested in finding some representation for an unknown function f_{trg}, called **target concept**—representable by some formula from a fixed, predefined formula class $\mathcal{R}$—, acquiring information about it via some protocol, defined by the given model. In the present dissertation two of these models are applied.

One such model is **query learning** (introduced by Angluin [10]), in which an oracle is assumed to answer (in constant time) questions of the learner via some query protocol. These questions are typically of the form of a **membership query**, querying the value of the target concept on some assignment, or an **equivalence query**, asking whether some formula, constructed by the learner is equivalent to the target concept. The **query complexity** of the class $\mathcal{R}$ is the (maximum of the) number of queries needed to ask by the learner depending on the **size** of f_{trg} (i.e., the length of the shortest formula in $\mathcal{R}$ for f_{trg}). A learning algorithm in this model is considered to be **efficient**, if both the quey complexity and the running time is polynomial (in the sum of the number of variables and the size of f_{trg}).

Another such model is the **mistake bounded model** (see e.g. [92]) which is defined in an on-line setting. In this model the learning proceeds in a sequence of rounds. In each round the learner receives first an instance of the domain (i.e., on which f_{trg} is defined) then produces a prediction of its classification, and finally receives a label (which, in a noise-free model is the correct classification—i.e., what f_{trg} evaluates on it). If the predicted classification and the received label disagree then the learner made a **mistake**. The **mistake bound** of the learning algorithm is the maximal number of mistakes, taken over all possible runs, (that is, sequences of instances), depending on the **size** of f_{trg}. A learning algorithm in this model is considered to be **efficient**, if both the quey complexity and the running time (in each round) is polynomial (in the sum of the number of variables and the size of f_{trg}).

Theory revision, as a special learning problem, assumes that the learner is not completely new to the given learning problem, hereby it has some initial hypotheses in the form of some formula that, albeit not equivalent to f_{trg}, but is thought to be a "good approximation" of it. A typical example is an initial version of an expert system provided by an expert, which needs to be refined using further examples or other information available. It is argued that this is a realistic requirement, as many complex concepts can only be hoped to be learned efficiently if a reasonably good initial approximation is available. Descriptions of theory revision systems are given, for example, in [82; 103; 107; 134; 135]. One of the first papers studying revision from a theoretical aspect is due to Mooney [100]. He assumed that the target can be obtained from the initial hypotheses by using **revision operators**, which are simple, predefined syntactic modifications, such as the deletion or the addition of a literal, and gave bounds for the the number of random examples needed in the PAC model for revision in terms of the number of these modifications necessary. Greiner [57] considered the computational complexity of hypothesis finding in a related framework.

The models for theory revision used in the present dissertation are extensions of

Mooney's approach to the query and the mistake bounded model. Actually, the models for theory revision differ from the corresponding learning models only in the efficiency criteria as follows: denoting the size of f_{trg} by s, and the minimal number of revision operators needed to apply on the initial hypotheses to obtain some representation for f_{trg} by $\hat{e}$, the number of queries asked (resp. the number of mistakes made) must be polynomial in $\hat{e}$ and in $\log m$ for an **efficient revision algorithm** [3]. (Note however that requirements set for the running time remains unchanged.)

For additional results on theory revision (not discussed in the present dissertation) the papers [50; 52; 53].

1.2 Characterization Results for Boolean Functions

Characterization results appear (and are applied) in several forms in mathematics and in computer science; like giving a semantic description for some object defined in a syntactic way (e.g. that a number, written in decimal form, is divisable by 5 if and only if its last digit is either 0 or 5), or to give an alternative syntactic description for some object defined in a syntactic way, and so on. Actually, it is one of the fundamental tools in the analysis of some mathematical object (like, say, a function, set, formula class, etc) to give an alternative description or representation for it, and work with that. It can, on one hand serve with more insight on the given object—which, in turn, can help solving the given problem—and, on the other hand (as is usual), it can provide more intriguing questions. A prominent examples for this is the Fourier transform of functions—i.e., to give an alternative representation for functions as a linear combination of some orthonormal system—, which is of invaluable importance, both in case of the real world applications and also on the theoretical level.

Characterization results are highly important for Boolean functions as well. A classical such result (see [71; 96]) is a semantic characterization of **Horn functions** (Boolean functions representable with **Horn formulas**—i.e., conjunctive normal form formulas in which every clause contains at most one unnegated variable). This result states that a function f is Horn if and only if for any pair of assignments on which f evaluates 1 it holds that f evaluates 1 also on their meet (i.e., componentwise $\wedge$). (This result is formulated in this dissertation as Theorem 10.2.) This, in turn, is used in the present dissertation to derive another characterization result involving Horn formulas.

Another classical characterization result (discovered independently several times—see [58; 74; 102]) considers **read-once functions** (Boolean functions representable with **read-once formulas**—i.e., formulas in which every variable occurs at most once). This result uses the notion of maxterms an minterms, which—focusing for simplicity only on monotone functions [4]— can be defined as follows: a minimal set of variables S is a **minterm** (resp. **maxterm**) of a monotone function f, if fixing the variables in

[3]An explanation for this choice of the efficiency criteria is given in Chapter 3.

[4]A Boolean function is said to be **monotone** if it is monotonically increasing in the usual sense.

S to 1 (resp. the variables in T to 0) forces f to take the value 1 (resp. 0). Then the characterization result states that a monotone Boolean function is representable by a read-once formula if and only if for arbitrary minterm S and maxterm T of it $|T \cap S| = 1$. A nice application of this result in learning theory is the learning algorithm constructed for read-once formulas in [13], which (although not applied, but still) is of special interest for us, as various learnability related properties of this class are analyzed in the present dissertation.

Finally note how central is the role of characterizing the extreme values and cases for some problems is in some fields. For instance extemal combinatorics (see e.g. [72]) is typically interested in questions of this sort; like that of determining the maximal number of prime implicants of Boolean functions. (A term t is an **implicant** of some Boolean function f, if any assignment saisfying t also satisfies f, meanwhile t is said to be a **prime implicant** of f if, in addition, this does not hold for any term obtained from t by removing some literals from it.) Considering this problem, it is known that a Boolean function on n variables can have at most $O\left(\frac{3^n}{\sqrt{n}}\right)$ prime implicants, and that there are functions with $\Omega\left(\frac{3^n}{n}\right)$ prime implicants (see, e.g., [31]), but the exact value for the maximal number of prime implicants is not known for general n. In the present dissertation a related problem is analyzed, which also takes into consideration the (minimal) number of terms in a DNF for a given function.

1.3 Results and the Structure of the Dissertation

The first part of the dissertation consists of results from theory revision, dealing with the revisability of some important formula class in various learning models. The second part consists of characterization results, some of which are related to some revision problem, meanwhile the rest is just interesting per se.

The first topic on theory revision in the dissertation is the revision of read-once functions (functions representable with formulas in which every variable occurs at most once) in the query model, discussed in Chapter 4. The importance of this formula class is rather theoretical, being a nontrivial subclass of Boolean formulas that is tractable from several different aspects, and has a nice semantic characterization [58; 74; 102]. As it has been shown by Angluin *et al.*, this class is also efficiently learnable with membership and equivalence queries [13] [5], it is thus natural to ask whether also an efficient revision algorithm exists for this class. This question is answered positively, but only for a restricted model which assumes that the function to be learned can be represented by a formula obtained from the initial one by deleting some parts of it. After that, the optimality of the algorithm is analyzed: a lower bound is shown for the query complexity of this class, that is of the same order of magnitude as the query complexity of the algorithm. Finally it is analyzed whether both types of the queries,

[5]What's more, read-twice functions are also efficiently learnable [104]—but read-thrice functions are not [2]. Here, read-twice (resp. read-thrice) functions (in accordance with the definition of read-once functions) are defined as functions that are representable with formulas in which every variable occurs at most twice (resp. three times).

used in the algorithm, are necessary, and it is shown that indeed, efficient revision is not possible using only one of the two types of queries.

As the next topic in theory revision, Chapter 5 considers Boolean threshold functions (i.e., functions representable by a set of variables R and a threshold θ, evaluting to 1 on exactly those assignments which assign 1 to at least θ of the variables in R). Threshold functions (although in a more general form) are famous for being the basic ingredient of neural networks and support vector machines—and has several other applications as well. For this class similar questions are asked as above. Again, a revision algorithm is presented in the query model, which, as shown, is an efficient algorithm for revising the class of threshold functions (in this case, however, no restriction is set on the model—i.e., both deletions and additions are allowed), having query complexity essentially optimal up to order of magnitude. Again it is shown that no efficient revision is possible for this class if one type of the queries gets banned. Finally it is shown that, somewhat surprisingly, `Winnow` [6]—a kind of multiplicative version of `Perceptron` being famous for learning some formula classes highly efficiently [7] using threshold representation—would not be a good choice for this task, as it would not work efficiently.

As a closure of the theory revision part, a subclass of the disjunctive normal form formulas, called **projective DNF**s, is considered in the mistake bounded model. For long it was one of the main open problems in computational learning theory, whether the class of DNFs is efficiently learnable. However recently it was proved that, unless RP = NP, the answer is no [5]. This motivates the search for subclasses of the DNFs which are efficiently learnable. The class of projective DNFs was introduced by Valiant [128] as a class suitable for projective learning—a notion motivated by certain biological considerations—; the general idea being that learning, similarly to other biological processes, should be carried out on multiple levels in a distributed manner. His construction consists of two levels. On the lower level simple learning algorithms are run, each concentrating on just a small part (or restriction) of the function to be learned. On the upper level another simple algorithm is run, which, on one hand, learns how to (re)combine the output of the algorithms on the lower level, and, on the other hand, it filters the information forwarded to these algorithms such that each one receives only that part of the information which is supposed to be relevant for it. Given this efficient algorithm for this class, it is an interesting question whether a natural extension it would behave as an efficient revision algorithm. After showing that the answer to this question is positive, some further, learning related features of the class are analyzed.

Being an appearently new class, projective DNFs provide several questions to be answered. One such that arose during examining this class was that a special subclass of it, called 1-projective DNFs (or 1-PDNFs for short) have shown some regularities in their syntax. (A DNF formula φ is 1-PDNF if every term t of it contains some literal ε such that $\varepsilon\varphi$ and t represent the same function.) Chapter 7 discusses this, and presents a characterization of this subclass that captures this regularity.

[6] More precisely a natural extension of it.
[7] More precisely in a so called attribute efficient manner.

Continuing the discussion of characterization results, the relation between the number of terms in a DNF, and the number of prime implicants of it is considered. Earlier results in computer science imply that if some DNF consists of K terms, then it has at most $2^K - 1$ prime implicants [31; 90; 97], and it has also been known previously, that this bound is sharp [88; 90; 97]. These results get completed in Chapter 8 in which a charactarization is given for DNFs that have as many prime implicants as this bound allows. This is shown by reducing the problem to the following problem: if in some DNF tautology each pair of terms conflict in exactly one variable (i.e., each pair is resolvable) then it posesses a tree-like structure (i.e., there is some variable v appearing in each term; there is some variable w appearing in each term that contains v negated, and there is some variable u in each term that contains v unnegated; and so on)—for which a new proof is presented.

The next characterization result considered is a generalization of the result, the previous problem (regarding the number of prime implicants of a DNF) is reduced to. More precisely it is shown in Chapter 9 that if in some DNF tautology each pair of terms conflict in at least one but at most two variables, then it also posesses a tree-like structure. However, further relaxing the bound given for the conflict of the terms to three, the above mentioned tree-like structure will not be automatic—as is demonstrated by an example. This problem is also a special case of a problem considered in [93], that, given a DNF tautology, the task is to construct a decision tree such that for each term of the DNF generated by it there is a term of the tautology that is a subterm of it. They have shown that even for some very simple DNFs this problem requires a decision tree with extremely big complexity; however the result presented in this chapter implies that for each DNF in the above mentioned restricted class there exists always some simple decision tree [8].

Finally, decomposable Horn formulas are discussed. Horn formulas, being an expressive class which also allows for polynomial time inference, and indeed is generally computationally tractable, play a central role in artificial intelligence and in computer science. The notion of decomposability comes from belief revision [9], a field interested in revising knowledge base in such a manner that satisfies some "reasonability" properties, that are typically formulated in the form of postulates. Decomposability was introduced for general logics in [41], where it was also shown to be equivalent to the existence of some revision operator satisfying the AGM postulates [4]—one of the most popular postulates used in belief revision. In Chapter 10 characterizations are given for the existence of a complement of a Horn function consequence of another Horn function, which in turn provides a complete description of decomposable Horn formulas. The characterizations lead to efficient algorithms for the construction of a complement whenever it exists (which is in contrast with a related, but somewhat more stringent complement notion of [60], the existence of which is occasionally NP-complete to decide). The result, as is purely combinatiorial, but was meant in [89] as a first step towards what is

[8]Actually the result states something stronger: for this restricted class basically the DNFs themselves can be considered as decision trees in some sense.

[9]Belief revision is related to theory revision (at least in it topic);thus—as a closure—the two main topics of the dissertation meet again.

referred to as "Horn-to-Horn belief revision": revision of Horn knowledge bases where the revised knowledge base is also required to be Horn; integrating hopefully efficient revision (the central notion in theory revision) and common sense reasoning (as a main goal in belief revision).

Chapter 2

General Definitions and Notations

When analyzing different representational classes it is often convenient (and sometimes maybe even unavoidable) to view formulas as functions and vice versa: to analyze a function by examining a formula representing it. Accordingly we frequently and freely switch between the semantical and the syntactical view. However, trying to keep the picture clear, we first discuss the two separately, and then discuss some connections of the two used heavily later on.

2.1 Syntax

$\mathcal{V} = \{v_1, v_2, v_3, \dots\}$ is the set of propositional variables in our universe, and for any integer n let $\mathcal{V}_n = \{v_1, v_2, v_3, \dots, v_n\}$. The **negation** of a variable $v \in \mathcal{V}$ is denoted $\overline{v}$. A **literal** is an unnegated or negated variable; unnegated variables are called **positive literals**; negated variables **negative literals**. The negation of the negative literal $\varepsilon = \overline{v}$, denoted $\overline{\varepsilon}$, is again the positive literal v.

A **Boolean formula** over variables $\mathcal{V}' \subseteq \mathcal{V}$ can be defined as the smallest subset of strings $\textsc{formulas}$ over $1, 0$, "$\vee$", "$\wedge$", ")", "(", "$^{-}$" and $\mathcal{V}'$ satisfying:

- $0, 1 \in \textsc{formulas}$ [1].
- Literals v and $\overline{v}$ are in $\textsc{formulas}$ for any $v \in \mathcal{V}'$.
- If $\varphi \in \textsc{formulas}$, then $\overline{\varphi} \in \textsc{formulas}$.
- If $\varphi_1, \dots, \varphi_k \in \textsc{formulas}$ and $k \geq 2$, then $\circ(\varphi_1, \dots, \varphi_k) \in \textsc{formulas}$, where $\circ$ is either $\vee$ or $\wedge$.

(In notation, for formulas greek lower case letters are used, usually φ and ψ, or sometimes χ.) Let $\mathrm{Var}(\varphi)$ (resp. $\mathrm{Lit}(\varphi)$) denote the set of variables (resp. set of literals) occuring in formula φ. For example if $\varphi = (v \vee \overline{w}) \wedge (w \vee (u \vee \overline{z}))$, then $\mathrm{Lit}(\varphi) = \{v, w, \overline{w}, u, \overline{z}\}$, meanwhile $\mathrm{Var}(\varphi) = \{v, w, u, z\}$, where $v, w, u, z \in \mathcal{V}$.

[1]For technical reasons, we extend the standard notion, which does not allow for constants in the leaves.

Besides Boolean formulas we also consider threshold formulas. A **threshold formula** is simply a pair (U, t), also denoted Th_U^t, where $U \subseteq \mathcal{V}$ and t is some non-negative integer.

Both Boolean and threshold formulas are often referred to simply as formulas.

2.1.1 Terms, Clauses, Special Formula Classes

A **term** (or **conjunction**) is a formula $\wedge(\varepsilon_1, \ldots, \varepsilon_k)$—often written in the form $\varepsilon_1 \wedge \cdots \wedge \varepsilon_k$—, where $\varepsilon_1, \ldots, \varepsilon_k$ are arbitrary literals. A k-**term** (or k-**conjunction**) is a conjunction of k literals. A **clause** (or **disjunction**) is the dual notion, where in the place of each $\wedge$ there is a $\vee$. Denote the empty conjunction (resp. empty disjunction) by $\top$ (resp. $\bot$). It is assumed that terms (resp. clauses) do not contain both a variable and its negation.

It is often convenient to treat clauses and terms as a set of literals; for example if $c = \overline{v_1} \vee v_3 \vee v_4$, then $\overline{v_1} \in c$ denotes that literal $\overline{v_1}$ appears in clause c, and if $t_1 = v_1 \wedge v_4$ and $t_2 = v_1 \wedge \overline{v_2} \wedge v_4 \wedge v_5$, then $t_1 = t_2 \setminus \{v_2, \overline{v_2}, v_5\}$ denotes that term t_1 can be obtained from t_2 by removing literal v_5 and removing variable v_2 with any orientation. (As it will always be clear from the text, wether the given formula is a clause or a term, this does not cause ambiguity.) Accordingly, the **size** of a term t, denoted by $|t|$, is the number of its literals, and some term t' is a **subterm** of t if $t' \subseteq t$ (which is obviously equivalent to $\mathrm{Lit}(t') \subseteq \mathrm{Lit}(t)$).

Terms t and t' **conflict** in variable v if v appears unnegated in one of them, and negated in the other. (In this case t and t' are also said to **collide**.) $t \otimes t'$ denotes the set of variables t and t' conflict in; thus $|t \otimes t'|$ is the number of conflicts between the two terms.

A **disjunctive normal form formula** (or **DNF** for short) is a disjunction of terms. A k-**DNF** is a DNF such that each of its terms contains at most k literals. A k-**term-DNF** is a DNF with at most k terms. Let k-**DNF**$_n$ (resp. k-**term-DNF**$_n$) denote the class of n-variable Boolean functions expressible as a k-DNF (resp. as a k-term-DNF). A **DDNF** or **disjoint DNF** is a DNF with pairwise conflicting terms. A DDNF formula has **conflict bound** d, if any two terms in it conflicts in at most two variables.

A **Horn clause** is a clause containing at most one positive literal. A **Horn formula** is a disjunction of Horn-clauses.

A **read-once** formula is a formula in which every variable occurs at most onnce.

As in the case of terms and clauses, sometimes DNFs are also treated as sets—in particular as a set of terms. Accordingly $t \in \varphi$ is used to denote that t is a term of the DNF φ.

A **Labeled Binary Tree** (or LBT) over variables in $\mathcal{V}' \subseteq \mathcal{V}$ is a rooted binary tree such that for each inner node the node itself and the edge leading to its right child are labelled by some $v \in \mathcal{V}'$, and the edge leading to its left child is labelled by $\overline{v}$. A **Decision Tree** (or DT) is an LBT that's leaves are labelled by 0 or 1.

2.2 Semantics

An **assignment** is a function $\mathbf{x} : \mathcal{V} \to \{0,1\}$, a **partial assignment** is partial function $\sigma : \mathcal{V} \hookrightarrow \{0,1\}$. In the latter case σ can also be considered as a function $\sigma : \mathrm{Dom}(\sigma) \to \{0,1\}$ where $\mathrm{Dom}(\sigma) := \sigma^{-1}(\{0,1\}) = \{v \in \mathcal{V} : v \text{ is assigned to some variable by } \sigma\}$ is the **domain** of σ. When $\sigma(v)$ appears in the text for some $v \in \mathcal{V}$, then it is implicitely understood that $\mathrm{Dom}(\sigma)$ contains v. The partial assignment with empty domain is denoted $()$.

When one focuses on a subset $\mathcal{V}''$ of the universe in scope (this often occurs when working with some (sub)formula φ, in which case $\mathcal{V}''$ is $\mathrm{Var}(\varphi)$), a partial assignment $\sigma : \mathcal{V}' \to \{0,1\}$ with $\mathcal{V}' \supseteq \mathcal{V}''$ can also be considered as an assignment. This is stressed in notation using bold face lower case Roman alphabet letters (usually $\mathbf{x}, \mathbf{y}, \mathbf{z}$, or sometimes $\mathbf{w}$ or $\mathbf{u}$) for these partial assignments, and to use lower case Greek letters (usually σ, or sometimes α) for those that leave some variables in $\mathcal{V}''$ unassigned. When $\mathcal{V}'$ is finite, say $\mathcal{V}' = \mathcal{V}_n$, σ can be written in the form $(v_1 \mapsto \sigma(v_1), \ldots, v_n \mapsto \sigma(v_n))$. For example if $\mathcal{V}' = \mathcal{V}_3$, and $\sigma(v_1) = 1$, $\sigma(v_2) = 0$ and $\sigma(v_3) = 1$, then $\sigma = (v_1 \mapsto 1, v_2 \mapsto 0, v_3 \mapsto 1)$. Also, for some $\mathcal{V}''' \subseteq \mathcal{V}$, let $\sigma|_{\mathcal{V}'''}$ denote the partial assignment that agrees with σ on $\mathcal{V}''' \cap \mathcal{V}'$, and leaves the rest of the variables unassigned.

$\mathbf{0}$ (resp. $\mathbf{1}$) denotes the assignment that assigns 0 (resp. 1) to each variable in scope, $\mathcal{V}''$, and for some $V \subseteq \mathcal{V}''$ let $\mathbf{1}_V$ denote the assignment assigning 1 to the variables in V and 0 to the variables in $\mathcal{V}'' \setminus V$.

Given two assignments $\mathbf{x}, \mathbf{y} : \mathcal{V}' \to \{0,1\}$, their **intersection** (or **meet**) is the assignment $\mathbf{x} \wedge \mathbf{y} : \mathcal{V}' \to \{0,1\}$ assigning $\mathbf{x}(v) \cdot \mathbf{y}(v)$ (i.e., the minimum of $\mathbf{x}(v)$ and $\mathbf{y}(v)$) to each variable $v \in \mathcal{V}'$. Also, the relation $\mathbf{x} \leq \mathbf{y}$ holds, if $\mathbf{x} = \mathbf{x} \wedge \mathbf{y}$, and $\mathbf{x} \lneq \mathbf{y}$ holds, if $\mathbf{x} \leq \mathbf{y}$ but $\mathbf{x} \neq \mathbf{y}$. Similarly to the meet, let the **join** of assignments $\mathbf{x}$ and $\mathbf{y}$ be the assignment $\mathbf{x} \vee \mathbf{y} : \mathcal{V}' \to \{0,1\}$ assigning $\mathbf{x}(v) + \mathbf{y}(v) - (\mathbf{x} \wedge \mathbf{y})(v)$ to variable $v \in \mathcal{V}'$ (i.e., assigning to each variable the maximum assigned to it by $\mathbf{x}$ and $\mathbf{y}$), and, finally, let $\mathbf{x} \otimes \mathbf{y} : \mathcal{V}' \to \{0,1\}$ assign $(\mathbf{x} \vee \mathbf{y})(v) - (\mathbf{x} \wedge \mathbf{y})(v)$ to variable $v \in \mathcal{V}'$.

Given some partial assignment σ and a variable $v \in \mathrm{Dom}(\sigma)$, the **component of** σ **corresponding to** v (or the v-**component** of σ, for short) is the partial assignment $\sigma|_{\{v\}}$. The v-component is said to be **on** (resp. **off**) in σ, if $\sigma(v) = 1$ (resp. $\sigma(v) = 0$). Let futhermore $\sigma^{[v]} = \sigma^{[\overline{v}]}$ be the partial assignment obtained from σ by flipping its v-component. For example $(v_1 \mapsto 1, v_2 \mapsto 0, v_3 \mapsto 1, v_4 \mapsto 0)^{[v_2]} = (v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 1, v_4 \mapsto 0)$ and also $(v_1 \mapsto 1, v_2 \mapsto 0, v_3 \mapsto 1, v_4 \mapsto 0)^{[\overline{v_2}]} = (v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 1, v_4 \mapsto 0)$.

The **Hamming distance** $\mathrm{distH}(\mathbf{x}, \mathbf{y})$ of assignments $\mathbf{x}$ and $\mathbf{y}$ is the number of variables on which $\mathbf{x}$ and $\mathbf{y}$ disagree. The **weight** of an assignment $\mathbf{x}$, denoted as $|\mathbf{x}|$, is the number of variables it assigns 1 to.

Given a set of variables $\mathcal{V}' \subseteq \mathcal{V}$, let $\mathcal{A}(\mathcal{V}')$ denote the set of assignments with domain $\mathcal{V}'$. Let furthermore $\mathcal{A}_n := \mathcal{A}(\mathcal{V}_n)$. A **Boolean function** f over variables $\mathcal{V}'$ is a zero-one valued function defined over the assignments with domain $\mathcal{V}'$—that is $f : \mathcal{A}(\mathcal{V}') \to \{0,1\}$. An n-**variable Boolean function** is a Boolean function over $\mathcal{A}_n$. Boolean functions will often be referred to simply as *functions*. In notation, plain lower case Roman alphabet letters (usually f, g or h) are used for Boolean function.

An assignment $\mathbf{x} \in \mathcal{A}(\mathcal{V}')$ is said **satisy** (resp. **falsify**) function f if $f(\mathbf{x}) = 1$ (resp. $f(\mathbf{x}) = 0$). The **truth set** of a function f is the set $\mathcal{T}(f) := \{\mathbf{x} \in \mathcal{A}(\mathcal{V}') : f(\mathbf{x}) = 1\}$, and let $\mathcal{F}(f) := \{\mathbf{x} \in \mathcal{A}(\mathcal{V}') : f(\mathbf{x}) = 0\}$. The function with truth set $\mathcal{A}(\mathcal{V}')$ (resp. $\emptyset$)—that is, which evaluates to 1 (resp. to 0) on each assignment—is denoted 1 (resp. 0). Finally note that a Boolean function over variables $\mathcal{V}' \subseteq \mathcal{V}$ can also be considered as a Boolean function over $\mathcal{V}''$ for any $\mathcal{V}' \subseteq \mathcal{V}'' \subseteq \mathcal{V}$.

For Boolean functions f and g write $g \leq f$ if every truth assignment satisfying g also satisfies f (i.e., if $\mathcal{T}(g) \subseteq \mathcal{T}(f)$). When this holds, g is said to **imply** f, or also that f is a **consequence** of g. If, in addition, there is a truth assignment $\mathbf{x}$ with $g(\mathbf{x}) = 0$ and $f(\mathbf{x}) = 1$, then g is said to **properly imply** f, or that f is a **proper consequence** of g, and denote it by $g \lneq f$.

A Boolean function f over variables $\mathcal{V}'$ is **monotone** if $\mathbf{x} \leq \mathbf{y}$ implies $f(\mathbf{x}) \leq f(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{A}(\mathcal{V}')$, it is **a-unate** for some $\mathbf{a} \in \mathcal{A}(\mathcal{V}')$, if $g(\mathbf{x}) = f(\mathbf{x} \otimes \mathbf{a})$ is monotone, and it is **unate** if it is $\mathbf{a}$-unate for some $\mathbf{a} \in \{0,1\}^n$.

Given (partial) assignments $\sigma_1 : \mathcal{V}' \to \{0,1\}$ and $\sigma_2 : \mathcal{V}'' \to \{0,1\}$ with $\mathcal{V}', \mathcal{V}'' \subseteq \mathcal{V}$, let $\sigma_1^{\sigma_2}$ be the (partial) assignment that agrees with σ_2 on $\mathcal{V}''$, with σ_1 on $\mathcal{V}' \setminus \mathcal{V}''$, and leaves the rest of the variables unassigned. When $\mathcal{V}'$ and $\mathcal{V}''$ are disjoint, then $\sigma_1^{\sigma_2}$ is sometimes written as (σ_1, σ_2). When this is the case, and $\sigma_1^{\sigma_2}$ is an input of some function f, or protocol MQ [2], then sometimes, instead of $f((\sigma_1, \sigma_2))$ or $\mathrm{MQ}((\sigma_1, \sigma_2))$, with a slight abuse of notation, simply $f(\sigma_1, \sigma_2)$, $\varphi(\sigma_1, \sigma_2)$ or $\mathrm{MQ}(\sigma_1, \sigma_2)$ is used.

2.3 Connecting Syntax and Semantics

Given a partial assignment $\sigma : \mathcal{V}' \to \{0,1\}$ and a Boolean formula φ over $\mathcal{V}$, let φ^σ be the formula obtained from φ by replacing each variable $v \in \mathrm{Var}(\varphi) \cap \mathcal{V}'$ with the value σ assignes to it. On the other hand, $\varphi(\sigma)$ is the formula obtained by iterating the following: if the current formula contains some subformula $\circ(\varphi_1, \ldots, \varphi_{i-1}, b, \varphi_{i-1}, \ldots, \varphi_\ell)$ for some $b \in \{0,1\}$, $\circ \in \{\wedge, \vee, ^-\}$, then replace it with

- 1, if $\circ$ is $\vee$ and $b = 1$, or if $\circ$ is "$^-$" and $b = 0$,
- 0, if $\circ$ is $\wedge$ and $b = 0$, or if $\circ$ is "$^-$" and $b = 1$,
- $\vee(\varphi_1, \ldots, \varphi_{i-1}, \varphi_{i-1}, \ldots, \varphi_\ell)$, if $\circ$ is $\vee$ and $b = 0$,
- $\wedge(\varphi_1, \ldots, \varphi_{i-1}, \varphi_{i-1}, \ldots, \varphi_\ell)$, if $\circ$ is $\wedge$ and $b = 1$,

as long as at least one of the above cases apply. Note that if σ is an assignment, then the resulting formula is either the 0 or the 1. Accordingly, for any formula φ there is a naturally associated function over variables $\mathrm{Var}(\varphi)$, mapping an assignment $\mathbf{x} \in \mathcal{A}(\mathrm{Var}(\varphi))$ to the appropriate constant $\varphi(\mathbf{x})$. Conversely, given some formula φ with an associated function f, we also say that φ **represents** f. Finally, define the empty conjunction, $\top$ (resp. the empty disjunction, $\bot$), to be always true (resp. false).

[2]See Chapter 3.

Given some threshold formula Th_U^t, and some (partial) assignment $\mathbf{x}$ with domain $\mathrm{Dom}(\mathbf{x}) \supseteq U$, let $\mathrm{Th}_U^t(\mathbf{x}) = 1$ if $\mathbf{x}$ assigns 1 to at least t of the variables in U, and let $\mathrm{Th}_U^t(\mathbf{x}) = 0$ otherwise. Accordingly, for any threshold formula there is a naturally associated function over variables U.

Two formulas φ and ψ are **equivalent**, denoted $\varphi \equiv \psi$, if they represent the same Boolean function. If we use some formula φ in a place where a function is expected, then φ will stand for the function represented by φ; accordingly the relations $\leq$ and $\lneq$ (i.e., the notions "implies" and "properly implies", resp. "consequence" and "proper consequence") can also be naturally extended for formulas. Now then, if a term t implies some function f than t is said to be an **implicant** of f. If, furthermore it also holds that deleting any literal from t results in a term that is not an implicant of f, then t is a **prime implicant** of f. On the other hand, if some clause c is a consequence of the Boolean function f, then c is called an **implicate** of f.

A term is **monotone** if it consists of unnegated variables. Given $\mathbf{a} \in \{0,1\}^n$, a term is $\mathbf{a}$-unate if the sign of every literal in it agrees with $\mathbf{a}$—that is, a literal is positive if and only if the corresponding component of $\mathbf{a}$ is 0. (Note that the above definitions coincide with the corresponding definitions for the associated functions.) For example, if $n = 3$ and $\mathbf{a} = 101$ then $\overline{v_1}\, v_2$ is $\mathbf{a}$-unate.

2.3.1 Vectors, Cubes and Subcubes

Let $\mathcal{V}' \subseteq \mathcal{V}$ be finite; for simplicity assume $\mathcal{V}' = \mathcal{V}_n$ for some n.

Note that (using the natural ordering of the variables in $\mathcal{V}$, where v_i is the i-th item in the order) assignments can be thought of as Boolean (or 0-1) vectors; accordingly $\mathcal{A}(\mathcal{V}')$ can be identified with the n-**dimensional cube**, $\{0,1\}^n$. Then, for example, the assignment $\sigma = (v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 1, v_4 \mapsto 0, v_5 \mapsto 1)$ can be written as $(1,1,1,0,1)$ or sometimes even as 11101. (Or maybe even using the exponential notation as $\sigma = 1^3 01$.)

A **subcube** (or simply **cube**) is any set of vectors that is of the form $\mathcal{T}(t)$ for some conjunction (i.e., term) t. For terms t_1, t_2, where $t_1 \not\equiv 0$, the following relations are equivalent:

- $t_1 \leq t_2$,
- $\mathcal{T}(t_1) \subseteq \mathcal{T}(t_2)$, and
- $\mathrm{Lit}(t_1) \supseteq \mathrm{Lit}(t_2)$, or in words: t_1 is **subsumed** by t_2.

For a literal ε, the ε **half cube** of $\mathcal{A}(\mathcal{V}')$ is the $(n-1)$-dimensional subcube formed by the vectors for which ε is true. If a term t is an implicant of a DNF $\varphi = t_1 \vee \cdots \vee t_k$, then we also say that φ is a **cover** of t, as the union of the cubes $\mathcal{T}(t_i)$ covers the cube $\mathcal{T}(t)$.

Proposition 2.1 *A set $A \subseteq \mathcal{A}(\mathcal{V}')$ is a cube if and only if for every $\mathbf{x}, \mathbf{y} \in A$ and every $\mathbf{z} \in \{0,1\}^n$ such that $\mathbf{x} \wedge \mathbf{y} \leq \mathbf{z} \leq \mathbf{x} \vee \mathbf{y}$, it also holds that $\mathbf{z} \in A$.*

Proof
The "only if" direction is easy to see.

The "if" direction follows by noting that the $\wedge$ and the $\vee$ of all the vectors in A is in A, and every vector between these two vectors is also in A. The conjunction of those literals to which value 1 is assigned by both of the above vectors is a term that is satisfied by exactly the vectors in A. $\square$

It follows, in particular, that if a cube contains two vectors with weights $w_1 < w_2$, then it also contains vectors of weight w for every $w_1 < w < w_2$.

Given $\mathbf{x}, \mathbf{y} \in \mathcal{A}(\mathcal{V}')$, the term corresponding to the smallest subcube containing both $\mathbf{x}$ and $\mathbf{y}$ is obtained by including every literal corresponding to components where $\mathbf{x}$ and $\mathbf{y}$ agree. For example, the smallest subcube in $\mathcal{A}_4$ containing both 1010 and 1100 is $v_1\overline{v_4}$.

Part I

Theory Revision Results

Chapter 3

Models and the Vapnik-Chervonenkis Dimension

In this chapter first a short description is given of the models used in the present dissertation. Although all the algorithms discussed in the later chapters are revision algorithms, the models used are variants of the appropriate models defined for learning. For this reason first the original variants are discussed shortly (in Section 3.1), and then the corresponding revision versions are defined (in Section 3.2). Note that, as the dissertation considers only Boolean functions and formulas, for simplicity the notions used are defined only for this case. (For a more general setting see e.g. [78].)

Finally, the Vapnik-Chervonenkis dimension is introduced [131]; a common tool used for proving lower bounds on the amount of information the learner needs to acquire about the target conecpt during the learning process.

3.1 Models for Learning

The first model discussed is PAC learning. Although it is not applied directly in the present dissertation, but this model (being the original model in computational learning theory [127]) gives the clearest (and at the same time: the rawest) picture of the general goals and nature of computational learning theory. For more on the relation of the models considered in this chapter and others see [63]. But let us first invoke from Chapter 1 the definition of learning given by Mitchell [99]:

> "A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

As mentioned, it is too general to be applicable for specific problems, but it sums up nicely what one has to specify, when formalizing a learning framework:

(a) the object for learning (i.e., what one wishes to learn),

(b) the method of acquiring information about it,

(c) some criteria for success, and

(d) (occasionally) some efficiency criteria.

A common feature of the models discussed below is that there assumed to be some fixed, predefined class of formulas $\mathcal{R}$ (e.g. the class of DNFs, or Horn formulas, or read-once formulas, etc) and some $f_{\mathrm{trg}} : \mathcal{A}(\mathcal{V}') \to \{0, 1\}$ representable by some formula in $\mathcal{R}$; the latter, which is referred to as the **target concept** [1], is unknown to the learner. The general task (thereby specifying (a)) is to find some representation for f_{trg} or for some approximation of it. Models requiring the former (i.e., to represent f_{trg} perfectly) are called **exact learning** models.

Another common feature is that the efficiency criteria builds on the **size of** f_{trg}, defined as the legth of the shortest formula in $\mathcal{R}$ representing f_{trg}.

3.1.1 Probably Approximately Correct Learning (PAC)

Recall that for the PAC model only a rough description is given, lacking the mere technicalities required by the exact definition, but sufficient to reveal the the general idea behind it.

Fix some distribution $\mathcal{D}$ over $\mathcal{A}(\mathcal{V}')$; this distribution, just like f_{trg}, is also unknown to the learner. Then, having access to randomly generated examples in the form $(\mathbf{X}_i, f_{\mathrm{trg}}(\mathbf{X}_i))$, $i = 1, 2, \ldots$, where $\mathbf{X}_1, \mathbf{X}_2, \ldots$ are independent and have distribution $\mathcal{D}$, the learner is required to, "with high probability" output some formula that "is a good approximation" of f_{trg} [2] —and, of course, to do all this efficiently in the complexity theoretic sense. It is easy to recognize the four items from the beginning of the chapter: (a) is f_{trg} , (b) is random data, (c) is that the probabilistic requirements are fulfilled and (d) is that the running time is polynomial in the size of the different parameters [3] (including the **size** of f_{trg}, defined as the length of the shortest formula in $\mathcal{R}$ representing it). Note that this bound for the *running time* also sets an **information theoretic** bound: it bounds the number of examples used.

3.1.2 Query Learning

In query learning (introduced by Angluin [10]) the learner collects information about the target concept through a query protocol (which thus specieis (b)), assuming the existence of an oracle that answers (in constant time) different type of questions of the learner. These questions are typically of the form of

- **membership query**, querying the value of f_{trg} on some assignment $\mathbf{x}$—asking for this information is usually denoted $\mathrm{MQ}(\mathbf{x})$—, or

[1]Note that a Boolean function (interpreting it as a membership function) can be thought of as a subset of the domain—or in other words as a **concept**.

[2]The conditions "with high probability" and "is a good approximation" are formulated in terms of the distribution $\mathcal{D}$.

[3]Basically, the size can be thought of as the number of bits needed to encode the different parameters, also including the size of a random example.

- **equivalence query**, querying for some **counterexample**: some assignment $\mathbf{x}$ on which f_{trg} and some formula φ, constructed by the learner, disagrees. (Counterexample $\mathbf{x}$ is called **positive**, if $f_{\mathrm{trg}}(\mathbf{x}) = 1$, or **negative**, if $f_{\mathrm{trg}}(\mathbf{x}) = 0$.) Asking for this information is usually denoted $\mathrm{EQ}(\varphi)$. Note that if such assignment does not exist (signaled by the oracle by returning $()$, the partial assignment with empty domain), then the learning process has come to an end: φ computes f_{trg}. The equivalence query $\mathrm{EQ}(\varphi)$ is **proper** if $\varphi \in \mathcal{R}$, otherwise it is **improper**.

Query learning is an exact learning model, thereby requiring that the learner learns f_{trg} *exactly* (again, this specifies (c)). Regarding (d), the efficiency criteria in this model, in accordance with the philosophy of the PAC model, is that the time required by the learner is bounded by a polynomial of the size of the parameters: the number of variables in focus and the length of the smallest formula in $\mathcal{R}$ for f_{trg}. Again, this bound for the running time also sets an information theoretic bound: it bounds the number of queries used.

3.1.3 Mistake Bounded Learning

In the mistake bounded model (see [92]) the learning proceeds in a sequence of rounds. In round r the learner receives an instance $\mathbf{x}_r$, and produces a prediction $\hat{y}_r$ of its classification. Then the learner receives a label y_r. (This actually completes the description of (b); in a noise-free model y_r is the correct classification of $\mathbf{x}_r$, that is, $y_r = f_{\mathrm{trg}}(\mathbf{x}_r)$.) If $\hat{y}_r \neq y_r$ then the learner made a **mistake**. The **mistake bound** of the learning algorithm is the maximal number of mistakes, taken over all possible runs—that is, sequences of instances. Regarding (d), the efficiency criteria is that both the number of mistakes and the time required by the learner in a round (but independently of the given round) can be bounded by a polynomial of the paremeters: the number of variables in focus and the length of the smallest formula in $\mathcal{R}$ for f_{trg}. (Here, the bound for the running time does not automatically set an information theoretic bound—i.e., for the number of mistakes committed—, this is why it had to be set directly.) As the model is thought of as an infinite process, it might not be that obvious, but this model is effectively an exact learning model, accordingly the success criteria (c) is that f_{trg} is learned exactly.

A mistake-bounded learning algorithm can be thought of as an equivalence query learning algorithm, where the equivalence queries correspond to the predictions at each stage of the algorithm. These queries are usually improper. Thus, proper equivalence and membership query algorithms and mistake-bounded algorithms are incomparable in general.

3.2 Models for Theory Revision

In theory revision the general task is the same as in learning: to construct some representation for the unknown target concept f_{trg} (in the dissertation only exact models are used for revision) acquiring information about it in the form specified by the given model. However it is also assumed that the learning does not start from sratch, and accordingly that the learner has some initial formula φ at hand. The general idea behind this (following the idea of Mooney [100]) is that applying some simple, predefined syntactic modifications (referred to as **revision oprations**) on φ one obtains a representation for f_{trg}. Thus, using φ, the learning requires less additional information about the target concept. On the other hand it is also apparent how strongly the learning task depends on the given initial hypotheses.

The revision operations can, in general, be either deletion or addition type. The definition of these operators may depend on the target class, but, in general, a **deletion operator** removes some literal occurence or some subformula from the given formula it is applied on, meanwhile an **addition operator** extends the formula with a literal occurence [4]. (Precise definitions for these operators for the different formula classes are given in the subsequent chapters.) The **revision distance** between the initial hypotheses φ and the target concept f_{trg}, denoted $\mathrm{dist}(\varphi, f_{\mathrm{trg}})$, is the minimal number of revision operations needed to transform φ to some formula representing f_{trg}. Note that the revision distance depends on the revision operators (differing in the different models!) and that it is *not* symmetric. Finally it should be mentioned that in some cases only one type of revision is considered. Accordingly one can differentiate between three cases: **deletions-only** (when only deletion operators are considered) [5], **additions-only** (when only addition operators are considered), and **general** (when both type of operators are considered).

To gain some intuition why approaching theory revision via the idea of revision operators is so appealing, note the following. Technically, the task of theory revision is to learn (i.e., construct some representation for) the "difference" of the initial hypotheses φ and the target concept f_{trg}—that is, to learn the set $\{\mathbf{x} : f_{\mathrm{trg}}(\mathbf{x}) \neq \varphi(\mathbf{x})\}$. To adopt the philosphy behind PAC learnability for this task, one has to assume then some representation class for the above set. However, there doesn't seem to be any natural, generally applicable method for this representation task that also fits the philosophy, other than to simply list the operators needed to apply on φ to obtain some representation ψ for f_{trg}.

The number of functions representable by some formula in $\mathcal{R}$ of size at most m is $2^{\Theta(m)}$ (unless using some wasteful representation, which we do not consider), thus, in general, to identify some formula of size m, one needs $\Theta(m)$ bits of information. This is reflected in/is in accordance with that in each learning model the information

[4]Basically, the addition operator is the inverse of the deletion operator which removes a single literal occurrence.

[5]As a technical detail, in this case it can happen that no representation of f_{trg} can be obtained from φ; in this case $\mathrm{dist}(\varphi, f_{\mathrm{trg}})$ can be defined to be infinite. However in the deletions-only case it is is always implicitely assumed that this is not the case. It should also be mentioned that there is a long history of studying this special case, presumably because of its greater tractability, in, and even before, the AI literature. Actually "deletions-only" corresponds to the "stuck-at" faults usually studied in diagnosing faulty circuits in the 1960s and 1970s (e.g., [81]) and, for instance, to the case where Koppel *et al.*proved the convergence of their empirical system for theory revision in the 1990s [82].

theoretic bound is at least linear (but maybe of some higher order polynomial) in the length of the smallest formula for f_{trg} [6].

In case of revision, the bound for the running time in the efficiency criteria still needs to be polynomial in the size of f_{trg} (and of course of φ as well), but the amount of information the learner needs depends on a completely different parameter: the amount of bits needed to encode f_{trg}, given φ. To encode the application of some revision operator one simply needs to encode where in the formula the revision operator is applied (and occasionally—in case of addition operators—also to encode some literal); thus, given φ, f_{trg} can be encoded using $O(\hat{e}(\log m + \log n))$ bits of information, where $\hat{e}$ denotes the revision distance between φ and f_{trg}, n the number of variables in use, and m the length of φ. Accordingly, in general, the infomation theoretic bound in the efficiency criteria for an efficient theory revision algorithm is typically polynomial in $\hat{e}(\log m + \log n)$.

Definition 3.1 (Theory revision in the query learning model) *Given some formula class $\mathcal{R}$, an algorithm is a* **revision algorithm** *for $\mathcal{R}$ with* **query complexity** *p, if, given any concept f_{trg}—called* **target concept**—*representable by some formula in $\mathcal{R}$, on input $\varphi \in \mathcal{R}$—called* **initial formula**—*the algorithm outputs some representation for f_{trg} using at most $p(\hat{e}, \log n)$ queries about f_{trg}, where $\hat{e} = \mathrm{dist}(\varphi, f_{\mathrm{trg}})$. The algorithm is said to be an* **efficient revision algorithm** *for $\mathcal{R}$, if p is a polynomial and the running time can also be bounded by a polynomial of the size of φ, the number of variables and $\hat{e}$. It is said that the query complexity of $\mathcal{R}$ is at least q, if any revision algorithm for $\mathcal{R}$ is of query complexity $\Omega(q)$.*

In theory revision equivalence queries are usually used to "detect" some flaw in the initial formula (i.e., to obtain some assignment on which the learner current hypotheses and the target concept disagrees), meanwhile membership queries (often applied in some kind of binary search) are usually used to "locate" the detected flaws (i.e., some position of the formula where some revision operator should be applied). It is often also interesting wether both types of queries are necessary for efficient revision of a given formula class. The thesis considers this problem for both formula classes for which efficient revision is provided in the query learning model.

Definition 3.2 (Theory revision in the mistake bounded model) *Given some formula class $\mathcal{R}$, an algorithm is a* **revision algorithm** *for $\mathcal{R}$ with* **mistake bound** *p, if, given any concept f_{trg}—called* **target concept**—*representable by some formula in $\mathcal{R}$, on input $\varphi \in \mathcal{R}$—called* **initial formula**—*the algorithm outputs some representation for f_{trg} making at most $p(\hat{e}, \log n)$ mistakes on instances classified by f_{trg}, where $\hat{e} = \mathrm{dist}(\varphi, f_{\mathrm{trg}})$. The algorithm is said to be an* **efficient revision algorithm** *for $\mathcal{R}$, if p is a polynomial and the running time in each round can also be bounded by a polynomial of the size of φ, the number of variables and $\hat{e}$.*

[6]Recall that both in query learning and in mistake bounded learning the information theoretic bound was allowed to depend also on n. However, results in **attribute efficient learning** (see e.g. [23; 27; 92]) suggest that this can often be omitted, and that the polynomial bound on the number of queries should allowed to depend only on the size of the targer concept; accordingly the dependence on n is not polynomial, only polylogarithmic (i.e., polynomial in $\log n$).

Finally it should be discussed how—or whether—theory revision results and learnability results imply each other. Obviously theory revision implies learnability (but only in the general case, allowing both addition and deletion opretors), but so far there are no satisfactorily general equivalence results for the other direction. And, in fact, it is not really expected to have one—as some results suggest:

- Read-once formulas (recall their deinition from Chapter 2) can be learned efficifently [13], and can also be revised efficiently in the deletions-only model (see Chapter 3), but considering the addition, it is not even clear what the right model should be.
- Horn-formulas (resp. monotone DNF formulas) can be learned efficiently [10; 12], but the revision problem of finding one deletion in an n-clause (resp. n-term) formula has query complexity $\Omega(n)$ [52; 53].
- Threshold functions can be learned using membership queries only, but in case of theory revision both query types are needed for the efficient revision (see Chapter 5).

This provides further motivation for researching the revisability of various important formula classes.

3.3 Vapnik-Chervonenkis Dimension

A common lower bound technique for the query complexity is to use the Vapnik-Chervonenkis dimension [131], which can be defined as follows.

Let $\mathcal{R}$ be a set of Boolean formulas over variables $\mathcal{V}'$. Some $Y \subseteq \mathcal{A}(\mathcal{V}')$ is said to be **shattered by** $\mathcal{R}$ if for any $Z \subseteq Y$ there is a $\varphi_Z \in \mathcal{C}$ such that

$$\varphi_Z(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in Z, \\ 0 & \text{if } \mathbf{x} \in Y \setminus Z. \end{cases}$$

Then $\text{VC-dim}(\mathcal{R}) := \max\{|Y| : Y \subseteq \mathcal{A}(\mathcal{V}')$ and Y is shattered by $\mathcal{R}\}$ is the **Vapnik-Chervonenkis dimension** of $\mathcal{R}$ [7].

Assume that the target concept is an arbitrary function that can be represented by some formula in $\mathcal{R}$. It is well known that in this setting any learning algorithm that uses only equivalence queries will ask at least $\text{VC-dim}(\mathcal{R})$ queries in the worst case. Furthermore (as is shown in [17] and in [94]), there is some universal constant $\alpha > 0$ such that even if the algorithm is allowed to ask both kind of queries (and even if the equivalence queries are improper), in the worst case it will ask at least $\alpha \cdot \text{VC-dim}(\mathcal{R})$ queries.

[7]Note that the Vapnik-Chervonenkis dimension is usually defined for some set of functions, and not formulas, however this approach seems to fit the presentation of the dissertation better.

Chapter 4

Read-once Formulas

Recall that a Boolean formula φ is a **read-once formula** (sometimes also called a μ-formula or a Boolean tree), if every variable has at most one occurrence in φ. Such a formula can be represented as a *binary* tree where the internal nodes are labeled with $\wedge$, $\vee$, and the negation and the leaves are labeled with *distinct* variables or the constants 0 or 1. (That is, for technical reasons—contrary to the general definition—we require that in read-once formulas all the $\vee$ and $\wedge$ operations are of arity two. Note, however that this does not mean the loss of generality; for example the formula $v \vee w \vee u$ can be represented as $\vee(v, \vee(w, u))$.) The internal nodes correspond to the subformulas.

Read-once formulas form a nontrivial class that is tractable from several different aspects, but slight extensions are already intractable. Boolean functions represented by read-once formulas have a combinatorial characterization [58; 74; 102], and certain read restrictions make CNF satisfiability easily decidable in polynomial time (see, e.g., [79]). It is interesting that the tractable cases for fault testing [81] and Horn theory revision [40; 82] are also related to read-once formulas.

Read-once formulas are efficiently learnable using equivalence and membership queries [13]. While read-twice DNF formulas are still efficiently learnable [104], for read-thrice DNF formulas there are negative results [2].

The main result in this chapter is the efficient revision algorithm for read-once formulas in the *query model* for the *deletions-only case*. Also lower bounds are provided showing that the algorithm is close to optimal.

4.1 Further Definitions and Notations

We call a subformula of φ **constant subformula** (more specifically; constant 0, resp. constant 1 subformula) if it computes a constant (constant 0, resp. constant 1) function. A constant subformula is **maximal constant subformula** if it is not the subformula of any constant subformula.

For technical reasons it is not the variables of some read-once formula φ that is of interest for us, rather the variables of φ that are not in some constant subformula of it. We call these variables the **relevant** variables of φ, and denote their set as $\mathrm{VarR}(\varphi)$.

Note that $\mathrm{VarR}(\varphi)$ can be determined in polynomial time for any read-once formula.

By the de Morgan rules, we may assume that negations are applied only to variables. As we consider read-once formulas only in the deletions-only model, and thus know the sign of each variable—we can replace the negated variables with new variables (keeping in mind that every truth assignment should be handled accordingly). Thus without loss of generality we can *assume that each variable is unnegated* (i.e., we use only $\wedge$ and $\vee$ in our read-once formulas). A Boolean function is a **read-once function** if it has an equivalent read-once formula.

4.1.1 Revision

For read-once formulas we only consider the deletions-only case (for the general case it is not even clear what the right model should be—recall Chapter 3). Note that for any formula obtained from some read-once formula φ by deleting some subformulas there is some equivalent formula obtained from φ by fixing some variables to 1, and some others to 0. Accordingly, the **revision operators** are the fixing of some variable to 0 or 1. Then the target concept is the associated function of $\psi = \varphi^{\hat{\sigma}}$ for some partial assignment $\hat{\sigma}$, where the initial hypotheses is φ, and the the **revision distance** of φ and ψ is $\mathrm{dist}(\varphi, \psi) := \min\{|\mathrm{Dom}(\sigma)| : \sigma \in \mathcal{A}(\mathcal{V}')$ such that $\psi \equiv \varphi^{\sigma}\}$, where $\mathcal{V}'$ is the universe in scope.

Note that this is in accordance with the general approach described in Chapter 3.

4.1.2 Sensitization

Our revision algorithm uses the technique of *path sensitization* from fault analysis in switching theory (see, e.g., Kohavi [81]). Let the initial formula be the monotone read-once formula

$$\varphi = (\varphi_1 \vee \varphi_2) \wedge \varphi_3 \,,$$

and let the target formula be

$$\psi = (\psi_1 \vee \psi_2) \wedge \psi_3 \,,$$

where ψ is obtained from φ by replacing certain variables by constants. Consider the partial truth assignment α that fixes all the variables in φ_2 to 0, and all the variables in φ_3 to 1. This fixing of the variables is called **sensitizing** φ_1 , and α is called the **sensitizing partial truth assignment** for φ_1. Put $\mathbf{x}_0 := \mathbf{0}^{\alpha}$ and $\mathbf{x}_1 := \mathbf{1}^{\alpha}$.

Asking the membership queries $\mathrm{MQ}(\mathbf{x}_0)$ and $\mathrm{MQ}(\mathbf{x}_1)$, there are three possibilities.

1. If $\mathrm{MQ}(\mathbf{x}_1) = 0$, then it must be the case that *either* $\psi_1(\mathbf{1}) = 0$, in which case ψ_1 is identically 0, *or* $\psi_3(\mathbf{1}) = 0$, in which case the whole target formula is identically 0.

2. If $\mathrm{MQ}(\mathbf{x}_0) = 1$, then it must be the case that *either* $\psi_1(\mathbf{0}) = 1$, in which case ψ_1 is identically 1, *or* $\psi_2(\mathbf{0}) = 1$, in which case ψ_2 is identically 1.

3. For the revision algorithm it is important to notice that we can also gain information in the third case, when $\mathrm{MQ}(\mathbf{x}_0) = 0$ and $\mathrm{MQ}(\mathbf{x}_1) = 1$. In this case we do not observe any "abnormality," but we can conclude that for every truth assignment $\mathbf{y} : \mathrm{VarR}(\psi_1) \to \{0, 1\}$ it holds that $\psi_1(\mathbf{y}) = \mathrm{MQ}(\mathbf{y}, \alpha)$. Thus we can simulate membership queries to the subformula ψ_1 by membership queries to the target concept, and this enables the revision algorithm to proceed by recursion. Also note that in this case it is still possible that $\psi_2(\mathbf{1}) = 0$ and/or $\psi_3(\mathbf{0}) = 1$.

Now we give the general definition of a sensitizing partial truth assignment. Let φ' be a subformula of φ that is not part of some constant subformula of it. Consider the binary tree representing φ, and let P be the path leading from the root of φ to the root of φ'. Then φ can be written as

$$\varphi = (\cdots(\varphi' \circ_r \varphi_r) \circ_{r-1} \cdots \circ_3 \varphi_3) \circ_2 \varphi_2) \circ_1 \varphi_1, \tag{4.1}$$

where $\varphi_1, \ldots, \varphi_r$ are the subformulas corresponding to the siblings of the nodes of P, and $\circ_1, \ldots, \circ_r$ are either $\wedge$ or $\vee$. In this representation we used the commutativity of $\wedge$ and $\vee$; in general φ' need not be a leftmost subformula of φ. Let ψ be obtained from φ by replacing certain variables by constants—that is, $\psi = \varphi^{\hat{\sigma}}$ for some partial assignment $\hat{\sigma}$. Then, as in (4.1), we can write ψ as

$$\psi = (\cdots(\psi' \circ_r \psi_r) \circ_{r-1} \cdots \circ_3 \psi_3) \circ_2 \psi_2) \circ_1 \psi_1. \tag{4.2}$$

where $\psi_i = \varphi_i^{\hat{\sigma}}$ for $i = 1, \ldots, r$. Subformula ψ' is called the **subformula corresponding to** φ'.

Definition 4.1 *Let φ be a read-once formula with subformula φ', and write φ as in Equation 4.1. Since φ is read-once, $\mathrm{VarR}(\varphi')$ and $\mathrm{VarR}(\varphi_i)$, $i = 1, \ldots, r$ form a partition of $\mathrm{VarR}(\varphi)$. Now let α be the partial truth assignment that assigns 1 (resp., 0) to every variable in $\mathrm{VarR}(\varphi_i)$ if $\circ_i$ is $\wedge$ (resp., $\vee$), for every $i = 1, \ldots, r$. Then α is called the* **partial truth assignment sensitizing** *φ'.*

Generalizing the remarks above, let α be the partial truth assignment sensitizing φ'. Form the truth assignments $\mathbf{x}_0 = \mathbf{0}^\alpha$ (resp. $\mathbf{x}_1 = \mathbf{1}^\alpha$) that extend α by assigning 0 (resp. 1) to the variables occurring in φ'. Now, if $\mathrm{MQ}(\mathbf{x}_1) = 0$, then it follows by the monotonicity of ψ that either ψ' or a subformula ψ_i such that $\circ_i = \wedge$ is constant 0. In this case, the whole subformula corresponding to $(\cdots(\psi' \circ_r \psi_r) \circ_{r-1} \cdots \circ_{i-1} \psi_{i-1}) \circ_i \psi_i$ in the target must be constant 0; thus this whole subformula can be deleted and replaced by 0. The case is similar when $\mathrm{MQ}(\mathbf{x}_0) = 1$. On the other hand, when $\mathrm{MQ}(\mathbf{x}_1) = 1$ and $\mathrm{MQ}(\mathbf{x}_0) = 0$, we can be sure that for any partial truth assignment $\mathbf{y}$ of the variables in ψ', we have $\psi'(\mathbf{y}) = \mathrm{MQ}(\mathbf{y}, \alpha)$. This means that ψ' is not part of a constant subformula of ψ. These remarks are summarized in the following lemma, which is used several times later on, sometimes without mentioning it explicitly.

Lemma 4.2 (a) *Let φ be the initial formula, φ' be a subformula of φ, let ψ, ψ' be the target formula, resp., its subformula corresponding to φ', and let α be the partial* *truth assignment sensitizing φ'. Then ψ' is part of a constant subformula of ψ if and only if* $\mathrm{MQ}(\mathbf{0}^\alpha) = 1$ *or* $\mathrm{MQ}(\mathbf{1}^\alpha) = 0$. *Otherwise* $\psi'(\mathbf{y}) = \mathrm{MQ}(\mathbf{y}, \alpha)$ *for every truth assignment* $\mathbf{y} : \mathrm{VarR}(\varphi') \to \{0, 1\}$.

(b) *If ψ' is a maximal constant subformula and $\circ_i$ is $\wedge$ (resp. $\vee$), then $\varphi_i(\mathbf{1}) = 1$ (resp. $\varphi_i(\mathbf{0}) = 0$) for every $i = 1, \dots, r$.*

In the rest of this section we formulate some useful properties of subformulas. Two subformulas are **siblings** if the corresponding nodes in the tree representation are siblings. The next lemma follows directly from the definitions.

Lemma 4.3 *Two maximal constant subformulas cannot be siblings.*

The revision algorithm proceeds by finding maximal constant subformulas, thus it is important to know that identifying these is sufficient for learning. That is, that the revised initial hypotheses, φ is equivalent to the target, $\psi = \varphi^{\hat{\sigma}}$, if the maximal constant subformulas of them are **identical**: correspond to the same inner nodes, and compute the same constant. For this, let us introduce the following notion. Partial assignments σ_1 and σ_2 are **equivalent** (with respect to some formula φ) if $\varphi^{\sigma_1} \equiv \varphi^{\sigma_2}$— or, equivalently, if $\varphi(\sigma_1) \equiv \varphi(\sigma_2)$.

Lemma 4.4 *(Partial) assignments σ_1 and σ_2 are equivalent for formula φ if and only if the maximal constant subformulas of φ^{σ_1} and φ^{σ_2} are identical.*

Proof
If the maximal constant subformulas are identical, then after replacing them with the corresponding constants, one obtains the same formula. Thus the "if" direction of the lemma holds. For the "only if" direction, assume that σ_1 and σ_2 are equivalent for φ, but the maximal constant subformulas are not identical. There are two cases. The first case is when there is a subformula φ_0 of φ that turns into a maximal constant subformula in both φ^{σ_1} and φ^{σ_2}, but $\varphi_0^{\sigma_1} \equiv 0$ and $\varphi_0^{\sigma_2} \equiv 1$. Let α be the partial truth assignment sensitizing φ_0. Then $(\varphi^{\sigma_1})(\mathbf{1}^\alpha) = 0$, while $(\varphi^{\sigma_2})(\mathbf{1}^\alpha) = 1$, contradicting the assumption that σ_1 and σ_2 are equivalent. In the second case there is a subformula which is maximal constant in one of φ^{σ_1} and φ^{σ_2}, but not for the other. Let φ_0 be a largest such subformula. We may assume *w.l.o.g.* that $\varphi_0^{\sigma_1}$ is a maximal constant subformula, which computes the constant 0, and $\varphi_0^{\sigma_2}$ is not part of a constant subformula. Then $\varphi^{\sigma_1}(\mathbf{1}^\alpha) = 0$ and $\varphi^{\sigma_2}(\mathbf{1}^\alpha) = 1$, again contradicting the assumption that σ_1 and σ_2 are equivalent. □

Corollary 4.5 *By finding a revision of the formula φ that has maximal constant subformulas identical to those of the target formula, we get a formula equivalent to the target formula.*

The following lemma can be proved by a simple algorithm that uses recursion on the structure of the formula φ.

Lemma 4.6 *Given a non-constant read-once formula φ and a constant c, one can find in polynomial time a partial assignment σ such that $\varphi^\sigma \equiv c$ and the number of variables in the domain of σ is minimal.*

Let φ be a read-once formula with subformula φ'. We say that φ' is an **approximately half-size subformula** of φ if $(1/3) \cdot |\mathrm{VarR}(\varphi)| \leq |\mathrm{VarR}(\varphi')| \leq (2/3) \cdot |\mathrm{VarR}(\varphi)|$. It is a standard fact that such a subformula exists (see, e.g., Wegener [132]). For example, any minimal subformula that contains at least one-third of the relevant variables has this property.

4.2 Revision Algorithm for Read-once Formulas

The main result of this section is for Algorithm `ReviseReadOnce` (Algorithm 1), which revises read-once formulas in the deletions-only model of revisions.

Algorithm `ReviseReadOnce` consists of a loop that checks whether the target has been found, and if not, calls `FindConstant`. (Recall that $()$ denotes the partial assignment with empty domain, and that receiving it for an equivalence query means that the queried formula is equivalent to the target formula.) In each call of `FindConstant` by `ReviseReadOnce`, a maximal constant subformula of the target formula ψ is identified along with a partial assignment that fixes this subformula to the appropriate constant value. The maximal constant subformula is then eliminated, thus the updated formula contains fewer variables. As the membership queries always refer to truth assignments to the original set of variables, the new membership queries have to assign some values to the eliminated variables as well. The construction implies that these variables are irrelevant, therefore their values can be arbitrary. In view of this, these variables will often be left out of consideration in the later steps.

Algorithm 1 Algorithm `ReviseReadOnce`(φ)

```
1: while $(\mathbf{x} := \mathrm{EQ}(\varphi)) \neq ()$ do
2:     $\sigma :=$ FindConstant($\varphi, \mathbf{x}$)
3:     $\varphi := \varphi^\sigma$
4: end while
```

`FindConstant`, displayed as Algorithm 2, is a recursive procedure, which takes a formula φ and a counterexample $\mathbf{x}$, and returns a partial assignment σ, which fixes a subformula to a constant c. It always holds that the subformula is a maximal constant subformula computing the constant c in any representation of the target concept [1]. `FindConstant` works recursively, always focusing on a faulty subformula (i.e., a subformula which contains some variable(s) replaced by a constant) of the previous level's formula. This subformula may never be a **proper** subformula of a constant

[1] In several places in the proof we will say that a property holds for any representation of the target concept. Notice that this must be true, as all the information used by the algorithm comes from membership and equivalence queries about the target, and the responses to such queries are independent of the particular representation.

subformula—that is, it is part of a constant subformula if and only if it itself is a maximal constant subformula. We assume this property holds at the beginning of every recursion level, and we maintain it as we go deeper in the recursion. This guarantees that we eventually find a maximal constant subformula. Once such a subformula is found, we use Lemma 4.6 to return an appropriate partial assignment fixing this subformula to constant c.

Algorithm 2 The procedure FindConstant($\varphi, \mathbf{x}$).

```
1:  if $\mathrm{MQ}(\mathbf{0}) == 1$ or $\mathrm{MQ}(\mathbf{1}) == 0$ then
2:      return $\sigma$ that fixes $\varphi$ to the appropriate constant
3:  end if
4:  Let $\varphi'$ be an approximately half-size subformula of $\varphi$
5:  Let $\alpha$ be the partial truth assignment sensitizing $\varphi'$
6:  if ( $c := \mathrm{MQ}(\mathbf{0}^\alpha) == \mathrm{MQ}(\mathbf{1}^\alpha)$ ) then
7:      return GrowFormula($\varphi, \varphi', c$)
8:  else
9:      Put $\mathbf{x}_1 := \mathbf{x}|_{\mathrm{VarR}(\varphi')}$ and $\mathbf{x}_{2,i} := \mathbf{x}|_{\mathrm{VarR}(\varphi_i)}$ for $i = 1, \ldots, r$
10:     if $\mathrm{MQ}(\mathbf{x}_1, \alpha) \neq \varphi'(\mathbf{x}_1)$ then
11:         return FindConstant($\varphi', \mathbf{x}_1$)          // look in $\varphi'$
12:     else
13:         $i :=$ FindFormula($\varphi, \varphi', \mathbf{x}$)
14:         return FindConstant($\varphi_i, \mathbf{x}_{2,i}$)
15:     end if
16: end if
```

As we go deeper in the recursion, we will need the ability to ask membership queries concerning only a subformula of the target. Therefore, when we go to a lower recursion level with a subformula χ of φ, we determine β, the partial truth assignment sensitizing χ. This way, whenever a need for a membership query arises on the lower level for a truth assignment $\mathbf{y} : \mathrm{VarR}(\chi) \to \{0, 1\}$, we need only ask $\mathrm{MQ}(\mathbf{y}, \beta)$. Recursion only occurs when $\mathrm{MQ}(\mathbf{0}^\beta) = 0$ and $\mathrm{MQ}(\mathbf{1}^\beta) = 1$, thus we can be sure that $\mathrm{MQ}(\mathbf{y}, \beta)$ is equal to the value of $\chi(\mathbf{y})$, where χ is the subformula of the target formula corresponding to χ (Lemma 4.2). From now on, when talking about membership queries, we always assume that this technique is used, even when, for simplicity, $\mathrm{MQ}(\mathbf{y})$ is written instead of $\mathrm{MQ}(\mathbf{y}, \beta)$.

Theorem 4.7 *Let φ be a read-once formula over $\mathcal{V}_n$, and the target formula be $\psi = \varphi^{\hat{\sigma}}$ for some partial assignment $\hat{\sigma}$. Then* ReviseReadOnce(φ)*, using at most $O(\hat{e} \log n)$ queries, outputs some partial assignment σ' such that $\psi \equiv \varphi^{\sigma'}$, where $\hat{e} = \mathrm{dist}(\varphi, \psi) = \min\{|\mathrm{Dom}(\sigma)| : \sigma \in \mathcal{A}_n$ such that $\psi \equiv \varphi^\sigma\}$.*

The theorem is an easy consequence of the following lemma. (Recall also Lemma 4.4.)

Lemma 4.8 *If $\varphi(\mathbf{x}) \neq \psi(\mathbf{x})$, then, using $p(\varphi, \mathbf{x}) = O(\log |\mathrm{VarR}(\varphi)|)$ queries, algorithm* FindConstant($\varphi, \mathbf{x}$) *returns a partial assignment $\sigma : \mathcal{V}' \to \{0, 1\}$ such that for* *some subformula $\tilde{\varphi}$ of φ with $\mathrm{VarR}(\tilde{\varphi}) \supseteq \mathcal{V}'$ it holds that the corresponding subformula $\hat{\psi}$ is a maximal constant subformula in ψ, and that $(\tilde{\varphi})^\sigma \equiv \hat{\psi}$. Furthermore, the cardinality of $\mathrm{Dom}(\sigma)$ is as small as possible.*

Proof

The proof of correctness uses an induction argument (based on the cardinality of $\mathrm{VarR}(\varphi)$), reflecting the recursive nature of the algorithm. The case when φ has at most one relevant variable, say v, is trivial: in this case ψ must be constant, which will be deteceted (using at most $p(\varphi, \mathbf{x}) = 2$ queries) in Line 3, and the algorithm simply returns some $\sigma = (v \mapsto c)$ for the appropriate $c \in \{0, 1\}$.

For the rest of the proof assume that $|\mathrm{VarR}(\varphi)| > 1$ and that the statement of the lemma holds for any formula having at most $(2/3)|\mathrm{VarR}(\varphi)|$ relevant variables. Let furthermore φ' be an approximately half-size subformula of φ. We also use the notations introduced in Equations (4.1) and (4.2), and Definition 4.1. Note furthermore that $|\mathrm{VarR}(\varphi_i)| \leq (2/3)|\mathrm{VarR}(\varphi)|$ for $i = 1, \ldots, r$.

If ψ is constant zero, or, equivalently, if $\mathrm{MQ}(\mathbf{0}) = 1$ or $\mathrm{MQ}(\mathbf{1}) = 0$ (see Lemma 4.2), then an appropriate output can be constructed as noted in Lemma 4.6. Again, $p(\varphi, \mathbf{x}) = 2$.

If ψ' is part of a constant subformula—that is, if $\mathrm{MQ}(\mathbf{0}^\alpha) = \mathrm{MQ}(\mathbf{1}^\alpha)$ (see Lemma 4.2)—, then (*Lines 6–7*) one only needs to find the maximal constant subformula it is in—or, in other words, to find the root of this maximal constant subformula on the path from the root of ψ to the root of ψ'. This can be carried out by procedure `GrowFormula` using $O(\log|\mathrm{VarR}(\varphi)|)$ queries (see Lemma 4.9 and the preceding description of the algorithm). It is thus also clear that $p(\varphi, \mathbf{x}) = O(\log|\mathrm{VarR}(\varphi)|)$.

For the subsequent arguments define $\mathbf{x}_1 := \mathbf{x}|_{\mathrm{VarR}(\varphi')}$ and $\mathbf{x}_{2,i} := \mathbf{x}|_{\mathrm{VarR}(\varphi_i)}$ for $i = 1, \ldots, r$.

If ψ' is not part of a constant subformula *and* $\mathrm{MQ}(\mathbf{x}_1, \alpha) \neq \varphi(\mathbf{x}_1, \alpha)$, then, by Lemma 4.2, $\varphi'(\mathbf{x}_1) \neq \psi'(\mathbf{x}_1)$, and thus ψ' contains a maximal constant subformula. By the induction hypthesis the call `FindConstant`$(\varphi', \mathbf{x}_1)$ (*Line 11*) will determine one such constant subformula $\hat{\psi}$, and return some partial assignment σ fulfilling the requirements of the lemma. Furthermore this call uses $p(\varphi', \mathbf{x}_1)$ queries, thus $p(\varphi, \mathbf{x}) = p(\varphi', \mathbf{x}_1) + 5$.

On the other hand, if ψ' is not part of a constant subformula, *but* $\mathrm{MQ}(\mathbf{x}_1, \alpha) = \varphi(\mathbf{x}_1, \alpha)$, then—as $\mathrm{MQ}(\mathbf{x}) \neq \varphi(\mathbf{x})$—it must hold that $\varphi_i(\mathbf{x}_{2,i}) \neq \psi_i(\mathbf{x}_{2,i})$ for some $1 \leq i \leq r$. Note that if some ψ_i is contained in some constant subformula, then this ψ_i *itself must be a maximal constant subformula*, as all other subformulas of ψ containing ψ_i also contain ψ', which is assumed not to be in a constant subformula. Thus if this i is known, a maximal constant subformula can be located by the recursive call `FindConstant`$(\varphi_i, \mathbf{x}_{2,i})$, using $p(\varphi_i, \mathbf{x}_{2,i})$ queries. Furthermore, `FindFormula` can be used to find such an index i using $\log\left(|\mathrm{VarR}(\varphi)|/|\mathrm{VarR}(\varphi_i)|\right) + 2$ queries (see Lemma 4.10 and the preceding description of the algorithm). Thus in this case $p(\varphi, \mathbf{x}) = p(\varphi_i, \mathbf{x}_{2,i}) + \log\left(|\mathrm{VarR}(\varphi)|/|\mathrm{VarR}(\varphi_i)|\right) + 7$.

This completes the analysis considering the correctness of the algorithm. In the rest of the proof we upper bound the number of queries made by `FindConstant`.

Denote by q the number of recursive calls, by m_i the number of relevant variables of the subformula in focus on the i-th level of recursion (thus $m_0 = \log|\mathrm{VarR}(\varphi)|$), and by p_i the number of queries made in the last $q-1$ level of recursion ($i = 0, 1, \dots, q$). First note that $m_i \le 2m_{i-1}/3$ for $i = 1, \dots, q$, thus $q = O(\log|\mathrm{VarR}(\varphi)|)$. Also note that $p_i \le p_{i-1} + 7 + \log(m_i/m_{i+1})$ for $i = 0, 1, \dots, q-1$ (i.e., on levels where some further recursive call was needed), meanwhile $p_q = O(\log m_q)$. Then

$$\begin{aligned} p(\varphi, \mathbf{x}) \le & \left(7 + \log\frac{m_0}{m_1}\right) + \dots + \left(7 + \log\frac{m_{q-1}}{m_q}\right) + p_q \\ = & O(\log m_0) + \log\frac{m_0 m_1 \cdots m_{q-1}}{m_1 m_2 \cdots m_q} + O(\log m_q) \\ = & O(\log m_0) \\ = & O(\log|\mathrm{VarR}(\varphi)|). \end{aligned}$$ □

Remark 4.1
Basically what happens in Lines 1–3 can be considered as part of the test in Line 6 and the binary search carried out by `GrowFormula` in Line7, but technically it seems to be easier to handle these cases separately. The same holds for Lines 10–11 and `FindFormula` in Line 13 too.

Remark 4.2
The analysis gets significantly more simple if, instead of the *weighted* binary search in `FindFormula`, one uses a simple binary search. However for that version of the algorithm only the query bound $O(\hat{e}\log^2 n)$ is proved (see [118]).

4.2.1 Algorithm `GrowFormula`

Now we give a description and analysis of algorithm `GrowFormula`. Throughout we use the notations of Equations (4.1) and (4.2), and Definition 4.1.

`GrowFormula` gets as input a monotone read-once formula φ, a subformula of it φ', and a constant c, such that $\mathrm{MQ}(\mathbf{0}^\alpha) = \mathrm{MQ}(\mathbf{1}^\alpha) = c$ (and thus $\mathrm{MQ}(\mathbf{y}, \alpha) = c$ for any partial truth assignment $\mathbf{y} : \mathrm{VarR}(\varphi') \to \{0,1\}$), where α is the partial truth assignment sensitizing φ'. It is also required that ψ is non-constant. Using $O(\log|\mathrm{VarR}(\varphi)|)$ membership queries it determines a subformula $\tilde{\varphi}$ containing φ' [2], such that the corresponding subformula in ψ is a *maximal* constant subformula (and is identical to the constant c). Finally `GrowFormula` outputs an appropriate partial assignment $\sigma : \mathrm{VarR}(\tilde{\varphi}) \hookrightarrow \{0,1\}$ such that $(\tilde{\varphi})^\sigma \equiv c$. In what follows we show how `GrowFormula` works.

Assume for simplicity that $c = 1$; the case $c = 0$ is dual. Let α_i for $i = 0, \dots, r$ be the partial truth assignment that is identical to α for variables in $\mathrm{VarR}(\varphi_1) \cup \dots \cup \mathrm{VarR}(\varphi_i)$, leaves the variables in $\mathrm{VarR}(\varphi')$ unassigned, and assigns 0 to all the variables

[2]Equivalently, as noted earlier, it determines the root of $\tilde{\varphi}$ on the path from the root of ψ to the root of ψ'.

in $\mathrm{VarR}(\varphi_{i+1}) \cup \cdots \cup \mathrm{VarR}(\varphi_r)$. Then $\mathbf{0} = \mathbf{0}^{\alpha_0} \leq \mathbf{0}^{\alpha_1} \leq \mathbf{0}^{\alpha_2} \leq \cdots \leq \mathbf{0}^{\alpha_r} = \mathbf{0}^{\alpha}$, and it holds that $\mathrm{MQ}(\mathbf{0}^{\alpha_0}) = 0$ and $\mathrm{MQ}(\mathbf{0}^{\alpha_r}) = 1$.

Asking membership queries $\mathrm{MQ}(\mathbf{0}^{\alpha_j})$, one can use binary search to find an i ($1 \leq i \leq r$) such that $\mathrm{MQ}(\mathbf{0}^{\alpha_{i-1}}) = 0$ and $\mathrm{MQ}(\mathbf{0}^{\alpha_i}) = 1$. The only difference between the truth assignments $\mathbf{0}^{\alpha_{i-1}}$ and $\mathbf{0}^{\alpha_i}$ is that the variables in $\mathrm{VarR}(\varphi_i)$ are off in $\mathbf{0}^{\alpha_{i-1}}$ and they may be on in $\mathbf{0}^{\alpha_i}$. In fact, they must be on, as otherwise $\mathbf{0}^{\alpha_{i-1}} = \mathbf{0}^{\alpha_i}$, contradicting $\mathrm{MQ}(\mathbf{0}^{\alpha_{i-1}}) \neq \mathrm{MQ}(\mathbf{0}^{\alpha_i})$. But (recalling the definition of the sensitizing partial truth assignment) $\mathbf{0}^{\alpha_{i-1}} \neq \mathbf{0}^{\alpha_i}$ also implies that $\circ_i$ is $\wedge$. Thus, on one hand, it must be the case that $\psi_i(\mathbf{0}) = 0$ and $\psi_i(\mathbf{1}) = 1$ in any representation of the target concept. On the other hand, it must be the case that the input to $\circ_i$ from its child on the path is equal to 1 in both cases. As the variables in this subformula are all set to 0, this subformula must compute the constant 1 function. The inputs $\mathbf{0}^{\alpha_{i-1}}$ and $\mathbf{0}^{\alpha_i}$ demonstrate that no larger subformula computes a constant function. Thus the subformula rooted at $\circ_{i-1}$ is a maximal constant subformula. Once a maximal constant subformula is found, one can simply apply Lemma 4.6 to construct an appropriate σ.

We have thus proved the following lemma.

Lemma 4.9 *If ψ is non-constant and $c = \mathrm{MQ}(\mathbf{0}^{\alpha}) = \mathrm{MQ}(\mathbf{1}^{\alpha})$, then it holds that* `GrowFormula`(φ, φ', c) *returns a partial assignment σ satisfying the requirements of Lemma 4.8, using $O(\log |\mathrm{VarR}(\varphi)|)$ queries.*

4.2.2 Algorithm `FindFormula`

Now we give a description and analysis of algorithm `FindFormula`. Throughout we use the notations of Equations (4.1) and (4.2), and Definition 4.1.

Assuming that ψ' is not part of some constant subformula of ψ, for $1 \leq i \leq r$ it holds that (as noted in the proof of Lemma 4.8) ψ_i is part of some constant subformula of ψ *only if* ψ_i itself is a *maximal* constant subformula of ψ. On the other hand, further assuming that $\psi(\mathbf{x}) \neq \varphi(\mathbf{x})$ but $\psi'(\mathbf{x}_1) = \varphi'(\mathbf{x}_1)$, it must thus hold that ψ has some maximal constant subformula in one of $\psi_1, \ldots \psi_r$. Given this, using $\log\big(|\mathrm{VarR}(\varphi)|/|\mathrm{VarR}(\varphi_i)|\big) + 2$ queries `FindFormula`$(\varphi, \varphi', \mathbf{x})$ outputs one such index i. In what follows we show how `FindFormula` works.

Put $y_{r+1} := z_{r+1} := \varphi'(\mathbf{x}_1)$ and for $i = 1, \ldots, r$ define y_i (resp. z_i) as

$$y_i = y_{i+1} \circ_i \varphi_i(\mathbf{x}_{2,i}), \text{ and } z_i = z_{i+1} \circ_i \psi_i(\mathbf{x}_{2,i}),$$

where $\mathbf{x}_1 := \mathbf{x}|_{\mathrm{VarR}(\varphi')}$ and $\mathbf{x}_{2,i} := \mathbf{x}|_{\mathrm{VarR}(\varphi_i)}$ for $i = 1, \ldots, r$. Then y_i (resp. z_i) is the value computed at $\circ_i$ in φ (resp. ψ) on the input vector $\mathbf{x}$, for $i = 1, \ldots, r$. Since (by the initial assumptions) $y_{r+1} = z_{r+1}$ and $y_1 \neq z_1$, there must be an i ($1 \leq i \leq r$) for which $y_{i+1} = z_{i+1}$ but $y_i \neq z_i$. The search for such an index i is done using a weighted binary search as follows. The y_i values can be computed using φ without any queries. For the computation of the z_i, put $\beta_{r+1} := \mathbf{x}_1$ and $\beta_j := (\beta_{j+1})^{\mathbf{x}_{2,j}}$ for $j = 1, \ldots, r$. Then (recalling that ψ_i is either a *maximal* constant subformula of ψ or is not part of a constant subformula of ψ) $z_i = \mathrm{MQ}(\alpha^{\beta_i})$.

Define for $j = 2, \ldots, r$ the **weight** of φ_j to be $w_j = |\mathrm{VarR}(\varphi_j)| + |\mathrm{VarR}(\varphi_{j-1})|$. In the binary search we use an interval $I = \{a, a+1, \ldots, b\}$. Initially $a = 2$ and $b = r$, as we already know y_1, z_1, y_{r+1} and z_{r+1}. For a given I let $s = \sum_{j \in I} w_j$. In each step we determine the index ℓ for which $\sum_{j=a}^{\ell-1} w_j < s/2 \leq \sum_{j=a}^{\ell} w_j$ (for this we don't need to ask any queries). We determine y_ℓ and z_ℓ (this can be done using one query). If $y_\ell \neq z_\ell$, then let $I = \{\ell+1, \ell+2, \ldots, b\}$, otherwise let $I = \{a, a+1, \ldots, \ell-1\}$. If I is nonempty, we compute s again, and continue the search. Otherwise the search is over, and if $y_\ell \neq z_\ell$, then ℓ is the i index we were looking for, otherwise it is $\ell - 1$.

To see that the above search uses the claimed number of queries, simply note that

- initially $s = \sum_{j \in I} w_j \leq (4/3)|\mathrm{VarR}(\varphi)|$, as the variables in φ' are not counted, whereas $|\mathrm{VarR}(\varphi')| \geq |\mathrm{VarR}(\varphi)|/3$
- in each step the value of the sum reduces to less than its half, and
- throughout the search $s \geq |\mathrm{VarR}(\varphi_i)|$, as even in the last step at least one of i and $i+1$ is in I,

so if t queries were made throughout the search, it holds that $|\mathrm{VarR}(\varphi_i)| \leq |\mathrm{VarR}(\varphi)| \cdot (4/3) \cdot (1/2^{t-1})$, implying

$$t \leq \log \frac{|\mathrm{VarR}(\varphi)|}{|\mathrm{VarR}(\varphi_i)|} + 3 - \log 3 < \log \frac{|\mathrm{VarR}(\varphi)|}{|\mathrm{VarR}(\varphi_i)|} + 2.$$

We have thus proved the following lemma.

Lemma 4.10 *If ψ' is not part of some constant subformula of ψ, and also $\psi'(\mathbf{x}_1) = \varphi'(\mathbf{x}_1)$, but $\psi(\mathbf{x}) \neq \varphi(\mathbf{x})$, then $\varphi_i(\mathbf{x}_{2,i}) \neq \psi_i(\mathbf{x}_{2,i})$ for some $1 \leq i \leq r$. Furthermore* `FindFormula`$(\varphi, \varphi', \mathbf{x})$*, using at most* $\log\big(|\mathrm{VarR}(\varphi)|/|\mathrm{VarR}(\varphi_i)|\big) + 2$ *queries, returns one such index i.*

4.3 Example Run of `ReviseReadOnce`

Here is a detailed example showing how the read-once revision algorithm works. Let $\mathcal{V}_9$ be the set of variables in focus, let the initial formula be

$$\varphi = ((v_1 \wedge v_2) \vee (v_3 \wedge v_4)) \wedge ((((v_5 \wedge v_6) \vee v_7) \wedge v_8) \vee v_9)$$

and let the target formula be $\psi := \varphi^\sigma$, where

$$\sigma = (v_3 \mapsto 1, v_5 \mapsto 0, v_6 \mapsto 0, v_8 \mapsto 0). \tag{4.3}$$

Thus the target concept is represented by the formula

$$\psi = ((v_1 \wedge v_2) \vee (1 \wedge v_4)) \wedge ((((0 \wedge 0) \vee v_7) \wedge 0) \vee v_9).$$

We start by asking the equivalence query $\mathrm{EQ}(\varphi)$. Let us assume that we receive the negative counterexample $\mathbf{x} = 110011110$. In Procedure `FindConstant`, the membership queries $\mathrm{MQ}(\mathbf{0}) = 0$ and $\mathrm{MQ}(\mathbf{1}) = 1$ bring us to Line 7. At this point we find an approximately half-size subformula , for example

$$\varphi' = (v_1 \wedge v_2) \vee (v_3 \wedge v_4).$$

The corresponding subformula of the target is $\psi' = (v_1 \wedge v_2) \vee (1 \wedge v_4)$.

Now we form the sensitizing truth assignment α for φ', which in this case simply sets all variables not in φ' to 1, and we ask membership queries for $(\mathbf{0}, \alpha)$ and for $(\mathbf{1}, \alpha)$. The answer is $\mathrm{MQ}(\mathbf{0}, \alpha) = 0$ and $\mathrm{MQ}(\mathbf{1}, \alpha) = 1$, and thus we continue on Line 12. We have $\mathbf{x}_1 = 1100$ and $\mathbf{x}_2 = 11110$. By asking the membership query $\mathrm{MQ}(\mathbf{x}_1, \alpha)$ we find that $\psi'(\mathbf{x}_1) = 1$. Knowing φ, we can determine without asking any queries that $\varphi'(\mathbf{x}_1) = 1$. As $\psi'(\mathbf{x}_1) = \varphi'(\mathbf{x}_1)$, it follows that the $\mathbf{x}_2$ part of the counterexample is responsible for the disagreement between $\varphi(\mathbf{x})$ and $\psi(\mathbf{x})$. In this particular case, the variables in $\mathbf{x}_2$ happen to induce a subformula of φ, and so `FindFormula` does not need to do anything. We substitute 1 for φ'. Then $\mathbf{x}_2 = 11110$ is a negative counterexample for the new target, which is the subformula ψ'' of the target corresponding to

$$\varphi'' = ((((v_5 \wedge v_6) \vee v_7) \wedge v_8) \vee v_9).$$

It is important to note that as $\psi''(\mathbf{y}) = \psi(\mathbf{x}_1, \mathbf{y})$, we can simulate membership queries to the new target by membership queries to the original target; thus we can continue the same procedure recursively.

As the subsequent iterations illustrate additional cases, we give further steps of the algorithm on the example. In the next call, which is `FindConstant`$(\varphi'', \mathbf{x}_2)$, we again get to Line 7. The next half size subformula can be $v_5 \wedge v_6$. The sensitizing truth assignment for this subformula is 010. Now, the membership queries to $(00, 010)$ and $(11, 010)$ both return 0, indicating that either $v_5 \wedge v_6$ or some subformula containing it is turned into the constant 0. Thus we call `GrowFormula`, which asks the additional membership queries $\mathrm{MQ}(11, 110) = 0$ and $\mathrm{MQ}(11, 111) = 1$. This shows that

$$(((v_5 \wedge v_6) \vee v_7) \wedge v_8)$$

is a maximal constant 0 subformula in φ''. No further recursive calls are needed, we only need to compute the minimal number of variables that, when turned to 0, make the subformula identically 0. This can be achieved by fixing the value of one single variable, that is, using the partial assignment $(v_8 \mapsto 0)$. Now we have completed one call of the procedure `FindConstant` by the main program.

The next call of `FindConstant` start with an equivalence query for the formula obtained above, that is,

$$\varphi''' = ((v_1 \wedge v_2) \vee (v_3 \wedge v_4)) \wedge v_9.$$

Let us assume that we receive the positive counterexample 000111111, which, restricted to the five variables in φ''', is 00011. We continue with the half size subformula $v_1 \wedge v_2$, which divides the counterexample into 00 and 011. The sensitizing partial truth assignment to the first half is 001. We find that $\mathrm{MQ}(00, 001) = 0$ and $\mathrm{MQ}(11, 001) = 1$, thus $v_1 \wedge v_2$ is not turned into a constant subformula. (Notice that our only membership oracle needs inputs from $\{0, 1\}^9$; fortunately, we may give any values to the "missing" variables.) The membership query $\mathrm{MQ}(00, 001) = 0$ tells us that the first half of the counterexample gives the same output in $v_1 \wedge v_2$ and in the corresponding subformula of the target. To recurse, we must find a subformula of φ''' that contains some constant subformula, but the three variables v_3, v_4, and v_9 do not induce a subformula of φ'''. This is achieved by the procedure `FindFormula`.

In this case we need consider only the two subformulas $v_3 \wedge v_4$ and v_9, though in general there could be $\Omega(n)$ such subformulas, necessitating the binary search performed by `FindFormula`. By definition, φ''' disagrees with the target on the counterexample, and we have just concluded that $v_1 \wedge v_2$ agrees with the counterexample. So, if subformula $(v_1 \wedge v_2) \vee (v_3 \wedge v_4)$ of φ''' disagrees with the corresponding subformula of the target, then the subformula containing a constant subformula must be $v_3 \wedge v_4$. Otherwise it is v_9. To test whether the subformula $(v_1 \wedge v_2) \vee (v_3 \wedge v_4)$ agrees with the target on the counterexample, we ask a membership query on an instance formed by setting v_1, v_2, v_3, and v_4 to the values they have in the counterexample, and setting the remaining variable (v_9) to the value it had in the sensitizing assignment for $v_1 \wedge v_2$. That, is we make the query $\mathrm{MQ}(00011) = 1$. Since $\varphi'''(00011) = 0$, which disagrees with the target, there must be a constant subformula in $v_3 \wedge v_4$, which is the input subformula for the next call to `FindConstant`.

That call will return the partial assignment $(v_3 \mapsto 1)$, and the next equivalence query to the formula

$$((v_1 \wedge v_2) \vee v_4) \wedge v_9$$

will finally identify the target concept. Notice that we have actually revised *fewer* variables than given in Equation 4.3. The number of variables revised is as small as possible for obtaining the target concept.

4.4 Lower Bounds on Revising Read-once Formulas

We prove a lower bound to the query complexity of revising read-once formulas by giving an example of an n-variable read-once formula, for which $\Omega(\hat{e}\log(n/\hat{e}))$ equivalence and membership queries are required to find a distance $\hat{e}$ revision. If $\hat{e} = O(n^{1-\varepsilon})$ for some fixed $\varepsilon > 0$, then this lower bound is of the same order of magnitude, as the upper bound provided by `ReviseReadOnce`. It is also shown that both types of queries are needed for efficient revision. There are n-variable read-once formulas for which at least $n/2$ equivalence queries are required in order to find a single revision. For membership queries we present an even stronger lower bound, which shows that at least

$n-\hat{e}$ membership queries may be necessary, if (instead of not using equivalence queries at all) one is allowed to use *fewer than* $\hat{e}$ equivalence queries. As `ReviseReadOnce` uses exactly $\hat{e}$ equivalence queries to find a distance $\hat{e}$ revision, this means that just by allowing one fewer equivalence query, the number of membership queries required becomes linear. Bshouty and Cleve and Bshouty et al. [28; 29] give somewhat related constructions and tradeoff results for different query types.

Our first two lower bounds are based on read-once formulas of the form $\bigvee(u_i \wedge w_i)$, using a Vapnik-Chervonenkis dimension, resp. an adversary argument, and the third lower bound uses an adversary argument for the n-variable disjunction.

Theorem 4.11 *The query complexity of revising read-once formulas in the deletions-only model is $\Omega(\hat{e}\log(n/\hat{e}))$, where n is the number of variables in the initial formula and $\hat{e}$ is the revision distance between the initial formula and the target formula.*

Proof
Let us assume that

$$n = 2\,m\,\hat{e}, \quad \text{where} \quad m = 2^t.$$

We use variables $u_{i,j}$ and $w_{i,j}$, where $1 \le i \le \hat{e}$ and $0 \le j \le m-1$. The initial formula is

$$\varphi_n = \bigvee_{i=1}^{\hat{e}} \bigvee_{j=0}^{m-1} (u_{i,j} \wedge w_{i,j}).$$

Assume the u and w variables be arranged in respective $\hat{e} \times m$ matrices called U and W, respectively. We look at the class of revisions of φ_n where in each row of the matrix U exactly one variable is fixed to 1. Let $\mathcal{R}_n$ denote the set of formulas that can be obtained this way.

Lemma 4.12 $\text{VC-dim}(\mathcal{R}_n) \ge \hat{e} \cdot t$.

Proof
For $1 \le k \le \hat{e}$ and $1 \le \ell \le t$ let

$$(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell})$$

be a truth assignment (to the variable pairs in $U \times W$) that consists of all 0's, with the exception of some positions in the k'th row of the W matrix: namely, those positions (k,j), where the ℓ'th bit of the binary representation of j is 1. Let the set of these assignments be S. We claim that S is shattered by $\mathcal{R}_n$.

Consider a subset $A \subseteq S$. For every k $(1 \le k \le \hat{e})$ let a_k be the t-bit number describing which truth assignments $(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell})$ belong to A. (That is, the ℓ'th bit of a_k is 1 iff $(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell}) \in A$.) We look at the revision φ_A for which it is the a_k'th variable which is fixed to 1 in row k of the matrix U.

It remains to show that this revision classifies S in the required manner. If $(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell}) \in A$, then bit ℓ of a_k is 1. By definition, $\mathbf{y}_{k,\ell}$ has a 1 at position (k, a_k). In φ_A, the variable u_{k,a_k} is fixed to 1. These observations imply that

$$\varphi_A(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell}) = 1.$$

On the other hand, if $(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell}) \notin A$, then bit ℓ of a_k is 0. The only 1 components of $(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell})$ are in row k of the W matrix: these are those positions (k, j), where the ℓ'th bit of the binary representation of j is 1. Position (k, a_k) is not one of those. Thus the corresponding u-variables are not fixed to 1 in φ_A, and as their value is 0, we get

$$\varphi_A(\mathbf{x}_{k,\ell}, \mathbf{y}_{k,\ell}) = 0.$$

□

By introducing dummy variables if n is not of the right form, we get

$$\text{VC-dim}(\mathcal{R}_n) \geq \hat{e} \left\lfloor \log \frac{n}{2\hat{e}} \right\rfloor .$$

The theorem now follows using the relation between the Vapnik-Chervonenkis dimension of a formula class and its query complexity (see Section 3.3). □

The number of formulas within revision distance $\hat{e}$ of a given read-once formula is at most $2^{\hat{e}} \cdot \binom{n}{\hat{e}}$. Thus if we allow equivalence queries which are not necessarily proper, then by using the standard halving algorithm [92] one can learn a revision using $\log \left(2^{\hat{e}} \cdot \binom{n}{\hat{e}}\right) = O(\hat{e} \log n)$ many equivalence queries. We now show that such a result is not possible if the queries are required to be proper.

Theorem 4.13 *The query complexity of revising read-once formulas in the deletions-only model with* proper *equivalence queries alone is at least $\lfloor n/2 \rfloor - 1$ (where n is the number of relevant variables in the initial formula), even when the revision distance is only one.*

Proof
Fix n, let $s = \lfloor n/2 \rfloor$, and let the initial formula be

$$\varphi = \bigvee_{i=1}^{s} (u_i \wedge w_i).$$

Let furthermore $\psi_i = \varphi^{(u_i \mapsto 1)}$ for $i = 1, \dots, s$, and set $\Psi = \{\psi_i : i = 1, \dots, s\}$. (Note that every element of Ψ is a potential target formula.)

Consider the following scenario. When the learner asks an equivalence query $\mathrm{EQ}(\varphi^\sigma)$ for some partial assignment σ, then the assignment returned is $\mathbf{x}$, where

- if $\varphi^\sigma \equiv \varphi$, then $\mathbf{x}$ is the positive counterexample $\mathbf{1}_{\{w_1,\dots,w_s\}}$. In this case Ψ remains unchanged.
- otherwise, if $\sigma(u_i) = 0$ or $\sigma(w_i) = 0$ for some $1 \leq i \leq s$, then $\mathbf{x}$ is the positive counterexample $\mathbf{0}^{(u_i \mapsto 1, w_i \mapsto 1)}$ Again, Ψ remains unchanged.
- otherwise, if $\sigma(w_i) = 1$ for some $1 \leq i \leq s$, then $\mathbf{x}$ is the negative counterexample $\mathbf{0}^{(u_i \mapsto 1)}$. Again, Ψ remains unchanged.

- otherwise, if for some $1 \leq i \leq s$ it holds that $\sigma(u_i) = 1$ but $\varphi_i \notin \Psi$, then $\mathbf{x}$ is the negative counterexample $\mathbf{0}^{(w_i \mapsto 1)}$. Again, Ψ remains unchanged.

- otherwise, if $|\Psi| > 1$, then $\mathbf{x}$ is the negative counterexample $\mathbf{0}^{(w_i \mapsto 1)}$ for some $1 \leq i \leq s$ such that $\sigma(u_i) = 1$. Also, remove ψ_i from Ψ.

- otherwise, that is, if $\sigma = (v_i \mapsto 1)$ for some $1 \leq i \leq s$ and $\Psi = \{\psi_i\}$, then $\mathbf{x} = ()$.

Note that during the whole process each element of the actual Ψ is consistent with all the previous informations, and that after each query $|\Psi|$ decreases by at most one. But, as the learning process cannot end as long as there are more than one non-equivalent hypotheses consistent with the previous informations, it follows that the learner must ask at least $\lfloor n/2 \rfloor - 1$ queries. □

Now we present a lower bound for the case when only membership queries are allowed. Actually, we consider a more general scenario, where the learner is allowed to ask a limited number of equivalence queries. In particular, we assume that the learner is told in advance that the target is at revision distance $\hat{e}$ from the initial theory, and the number of equivalence queries allowed is at most $\hat{e} - 1$.

Theorem 4.14 *Denote the revision distance between the initial formula and the target formula by $\hat{e}$, and assume that the learner is allowed to ask arbitrarily many membership queries, but only at most $\hat{e} - 1$ equivalence queries. Under this restriction the query complexity of revising read-once formulas in the deletions-only model is at least $n - \hat{e}$, where n is the number of relevant variables in the initial formula.*

Proof

Let the initial formula be $\varphi = \bigvee_{v \in \mathcal{V}_n} v$, and set initially $D = R = \emptyset$ and $U = \mathcal{V}_n$. (D stands for *deleted*, R stands for *relevant* and U stands for *uncertain*.)

Consider the following scenario. When the learner asks an equivalence query $\mathrm{EQ}(\varphi^\sigma)$ for some partial assignment σ, then the assignment returned is $\mathbf{x}$, where

- if it holds that $U = \emptyset$ and $\varphi^\sigma \equiv \bigvee_{v \in R} v$, then $\mathbf{x} = ()$.

- otherwise, if φ^σ is identically 1 (resp., 0), then $\mathbf{x}$ is the negative (resp., positive) counterexample $\mathbf{0}$ (resp., $\mathbf{1}$). In this case the sets are not changed.

- otherwise, if $U \setminus \mathrm{Dom}(\sigma) \neq \emptyset$, then $\mathbf{x}$ is the negative counterexample $\mathbf{0}^{(v \mapsto 1)}$ for some $v \in U \setminus \mathrm{Dom}(\sigma)$. In this case move v from U to D.

- otherwise $\mathbf{x}$ is the positive counterexample $\mathbf{1}_U$. Again, the sets are not changed.

When the learner asks a membership query $\mathrm{MQ}(\mathbf{x})$ for some assignment $\mathbf{x}$, then the answer is

- "1" if $\mathbf{x}(v) = 1$ for some $v \in R$. In this case the sets are not changed.

- "1" otherwise, if $\mathbf{x}(v) = 1$ for some $v \in U$. In this case one such v is moved from U to R. Furthermore, if now $|U \cup D| = \hat{e}$, then move the rest of the variables of U to D.
- "0" otherwise, and the sets are not changed.

Note that D can only increase after an equivalence query, and even then only by one. Thus, according to the assumptions of the theorem, the cardinality of D will always be less then $\hat{e}$. It also holds that $|U \cup D|$ does not change after an equivalence query, and decreases by at most one after a membership query, as long as $|D \cup U| > \hat{e}$. Finally note that during the whole process for each $V \subseteq U \cup R$ of cardinality $\hat{e}$ it holds that $\bigvee_{v \in \mathcal{V}_n \setminus V} v$ is consistent with all the previous informations. But, as the learning process cannot end as long as there are more than one non-equivalent hypotheses consistent with the previous informations, it follows that the learner must ask at least $n - \hat{e}$ membership queries. $\square$

4.5 Concluding Remarks

All the results presented in this chapter—unless noted otherwise—appeared in the paper [52], co-authored by the author of the present dissertation.

Chapter 5

Threshold Formulas

Recall that on assignment $\mathbf{x}$ threshold formula Th_U^t evaluates 1 if $\mathbf{x}$ assigns 1 to at least t variables in U, otherwise it evaluates to 0. A **threshold function** is a Boolean function that can be represented with some threshold formula. Functions of this type are also called Boolean threshold functions and zero-one threshold functions, in order to distinguish them from the more general kind of threshold functions, where instead of simply counting the number of variables in U assigned to 1, one associates weights to variables, and sums the weights of the components that are on. (For example such a threshold function is applied in Algorithm `RevWinn` in Section 6.2.) However, as in this chapter only the former class is considered, throughout this restricted class is referred to as threshold functions.

Threshold functions (especially in the wider, non-Boolean sense) form a much studied concept class in computational learning theory. They are also applied in many learning related results (see e.g. [92; 126; 129]). Hegedűs [64] gave $\Theta(n)$ upper and lower bounds (assuming that $\mathcal{V}_n$ is the set of variables in focus) for the number of queries needed to learn threshold functions in the query model; the algorithm uses only membership queries.

In this chapter an efficient revision algorithm is presented for the class of threshold function in the *query model* for the *general case* (also allowing the modification of the threshold). Additionally, some negative results are presented showing, for instance, that threshold functions cannot be revised efficiently from either type of query alone.

5.1 Further Definitions and Notations

For simplicity assume throughout the chapter that $\mathcal{V}_n$ is the set of variables in focus.

For some threshold function Th_U^t the variables in U (resp., in $\mathcal{V}_n \setminus U$) are the **relevant** (resp., **irrelevant**) variables of Th_U^t. Note that for every non-constant threshold function its set of relevant variables and its threshold are well defined, thus every non-constant threshold function has a unique representation. We say that a set $S \subseteq \mathcal{V}_n$ is a **positive** (resp., **negative**) set for Th_U^t if it evaluates to 1 (resp. to 0) on $\mathbf{1}_S$.

A set $S \subseteq \mathcal{V}_n$ is **maximal negative** (or **critical**) for threshold function Th_U^t if

$|S \cap U| = t - 1$; and **minimal positive** for Th_U^t if $|S \cap U| = t$.

Given the above, we can state the following proposition which we use implicitly throughout:

Proposition 5.1 *If S is maximal negative for $\psi = \mathrm{Th}_U^t$, then for every $Z \subseteq \mathcal{V}_n \setminus S$ it holds that Z contains at least one variable in U (i.e., relevant variable of ψ) if and only if $\mathrm{MQ}(\mathbf{1}_{S \cup Z}) = 1$.*

5.1.1 Revision

In the case of threshold functions the general model is used, where a **deletion operator** is the deletion of a relevant variable and an **addition operator** is the addition of a new relevant variable, and, additionally, it is also allowed the *modify the threshold*. More precisely, the modification of the threshold by any amount is considered to be a single operation (as opposed to changing it by one); as for the algorithm upper bounds are proved, this only makes the results stronger. Thus the **revision distance** is defined as

$$\operatorname{dist}\left(\mathrm{Th}_U^t, \mathrm{Th}_R^\theta\right) = \begin{cases} |U \setminus R| + |R \setminus U| + 1, & \text{if } t = \theta, \\ |U \setminus R| + |R \setminus U|, & \text{otherwise.} \end{cases}$$

Thus, for example, $\operatorname{dist}\left(\mathrm{Th}^1_{\{v_1,v_2,v_4\}}, \mathrm{Th}^3_{\{v_1,v_2,v_3,v_5\}}\right) = 4$.

Note that this is in accordance with the general approach described in Chapter 3.

5.2 Revision Algorithm for Threshold Functions

We present a threshold revision algorithm `ReviseThreshold`. The overall revision algorithm is given as Algorithm 3, using the procedures described in Algorithms 5 and 6. Throughout this section, let the initial function be $\varphi = \mathrm{Th}_U^t$ and the target function be $\psi = \mathrm{Th}_R^\theta$. Algorithm `ReviseThreshold` has three main stages. First it identifies all the variables that are irrelevant in φ but relevant in ψ (Algorithm `FindAdditions`). Then it identifies all the variables that are relevant in φ but irrelevant in ψ (Algorithm `FindDeletions`). Finally, it determines the target threshold. (In the pseudocode this third step is built into Algorithm `FindDeletions` as the last iteration, after the set of relevant variables of the target function is identified.)

A sample run of the algorithm is given in Section 5.3.

Algorithm 3 The procedure `ReviseThreshold`(φ), where $\varphi = \mathrm{Th}_U^t$.

```
1: Use 2 MQ's to determine if $\psi \equiv c$ for some $c \in \{0,1\}$; if so return c
2: $V :=$ FindAdditions($U$)
3: $\psi :=$ FindDeletions($U \cup V$)
4: return $\psi$
```

Before getting into further details, we need to point out an additional subroutine. Our revision algorithm frequently uses a kind of binary search, presented as Algorithm 4.

The starting points of the binary search are two sets, a negative one, N and a positive one, P such that $N \subseteq P$. The algorithm returns two items: the first is a set of variables that, when added to N, make a positive set; the second is a variable that, when removed from this positive set, turns it into a negative one.

Algorithm 4 `BinarySearch`(N, P).

```
Require: $\mathrm{MQ}(\mathbf{1}_N) = 0$ and $\mathrm{MQ}(\mathbf{1}_P) = 1$ and $N \subseteq P$
1: $N_0 := N$
2: while $|P \setminus N| > 1$ do
3:    Partition $P \setminus N$ into approximately equal-size sets $D_1$ and $D_2$.
4:    Put $M := N \cup D_1$
5:    if $\mathrm{MQ}(\mathbf{1}_M) == 0$ then
6:       $N := M$
7:    else
8:       $P := M$
9:    end if
10: end while
11: Let $v$ be the one variable in $P \setminus N$
12: return $(P \setminus N_0, v)$
```

First we analyze algorithm `FindAdditions` (Algorithm 5), which is responsible for finding all missing relevant variables.

Lemma 5.2 *Let R be the relevant variables of the nonconstant target function. If Algorithm* `FindAdditions` *is called with input $U \subseteq \mathcal{V}_n$, then it returns $R \setminus U$, using $O(|R \setminus U| \log n)$ queries.*

Proof
The algorithm stores the uncertain but potentially relevant variables in the set $\textsc{Potentials}$ (thus $\textsc{Potentials}$ is initially set to $\mathcal{V}_n \setminus U$). The procedure first determines a set $\textsc{Base} \subseteq U$ such that $\textsc{Base}$ is negative, and $\textsc{Base} \cup \textsc{Potentials}$ is positive (unless $\textsc{Potentials}$ contains no relevant variables—in which case there are no new relevant variables used by ψ, so we quit in Line 8).

Then the search for new relevant variables starts. `BinarySearch`$(\textsc{Base}, \textsc{Base} \cup \textsc{Potentials})$ is used repeatedly to find one relevant variable, and then remove this variable from $\textsc{Potentials}$. After removing a certain number of relevant variables from $\textsc{Potentials}$, the instance $\textsc{Base}\cup\textsc{Potentials}$ must become minimal positive. After reaching this point, we do not only remove any newly found relevant variables from $\textsc{Potentials}$, but we also add them to the set $\textsc{Base}$. From this point on, it holds that $|(\textsc{Base}\cup\textsc{Potentials})\cap R| = \theta$. Thus the indicator that the last relevant variable has been removed from $\textsc{Potentials}$ is that $\textsc{Base}$ becomes positive ($\mathrm{MQ}(\mathbf{1}_{\textsc{Base}}) = 1$).

As `BinarySearch` always uses at most $\lceil \log_2 n \rceil$ membership queries per call, and one addition requires one call to `BinarySearch` and at most two other membership queries are made initially, the stated query complexity follows. □

Now we turn to the discussion of procedure `FindDeletions` (Algorithm 6), which finds all the irrelevant variables that appear in the initial hypotheses. The procedure

Algorithm 5 The procedure `FindAdditions`(U)

```
Require: the target function is not constant
 1: $\textsc{Potentials} := \mathcal{V}_n \setminus U$
 2: if $\mathrm{MQ}(\mathbf{1}_U) == 0$ then
 3:     $\textsc{Base} := U$
 4: else
 5:     $(\textsc{Base}, v) :=$ BinarySearch$(\emptyset, U)$
 6:     $\textsc{Base} := \textsc{Base} \setminus \{v\}$
 7:     if $\mathrm{MQ}(\mathbf{1}_{\textsc{Base}\cup\textsc{Potentials}}) == 0$ then
 8:         return $\emptyset$
 9:     end if
10: end if
11: $\textsc{NewRelevants} := \emptyset$
12: repeat
13:     $(V, v) :=$ BinarySearch$(\textsc{Base}, \textsc{Base} \cup \textsc{Potentials})$
14:     $\textsc{NewRelevants} := \textsc{NewRelevants} \cup \{v\}$
15:     $\textsc{Potentials} := \textsc{Potentials} \setminus \{v\}$
16:     if $\mathrm{MQ}(\mathbf{1}_{\textsc{Base}\cup\textsc{Potentials}}) == 0$ then
17:         $\textsc{Base} := \textsc{Base} \cup \{v\}$
18:     end if
19: until $\mathrm{MQ}(\mathbf{1}_{\textsc{Base}}) == 1$
20: return $\textsc{NewRelevants}$
```

uses a function called `MakeEven`, presented as Algorithm 7. `MakeEven` makes at most two queries; its main task is to move variables around to ensure needed conditions, mostly parity, on certain sets. A more detailed prose description of its behavior is given in the proof of Lemma 5.3.

Lemma 5.3 *If the target function $\psi = \mathrm{Th}_R^\theta$ is not constant and if $R \subseteq H \subseteq \mathcal{V}_n$, then if Algorithm* `FindDeletions` *is called with input H, it returns ψ, using $O(|H \setminus R| \log n)$ queries.*

Proof

First consider the case where no variables need to be deleted from H. If the threshold is either 1 or $|H|$, this will be found by one of the two initial equivalence queries to those two threshold functions. (Recall that () denotes the partial assignment with empty domain, and that receiving it for an equivalence query means that the queried formula is equivalent to the target formula.) If the threshold is some value in between, then it will be found by a binary search over threshold values carried out by the first while loop. Then the correct threshold function is returned (at Line 12).

Otherwise, there are some variables that need to be deleted. In this case, our short-term goal is to find two sets of variables N and P such that

$$|N| \geq |P|, \text{ and } N \text{ is negative and } P \text{ is positive for } \mathrm{Th}_R^\theta\ . \tag{5.1}$$

The two initial equivalence queries must have assigned P to be a positive counterexample to Th_H^1 and N to be a negative counterexample to $\mathrm{Th}_H^{|H|}$. In the binary

Algorithm 6 The procedure `FindDeletions`(H)

```
Require: $R \subseteq H$ ($R =$ relevant variables in target)
 1: if $\left(\mathbf{x}_P := \mathrm{EQ}\left(\mathrm{Th}_H^{|H|}\right)\right) == ()$ then
 2:    return $\mathrm{Th}_H^{|H|}$
 3: end if
 4: if $\left(\mathbf{x}_N := \mathrm{EQ}\left(\mathrm{Th}_H^{1}\right)\right) == ()$ then
 5:    return $\mathrm{Th}_H^{1}$
 6: end if
 7: $P := \{v \in H : \mathbf{x}_P(v) = 1\}$, $N := \{v \in H : \mathbf{x}_N(v) = 1\}$,
 8: $\ell := 1; u := |H|$
 9: while $u > \ell + 1$ do
10:    $m := \lceil (u + \ell)/2 \rceil$
11:    if $(\mathbf{x} := \mathrm{EQ}(\mathrm{Th}_H^m)) == ()$ then
12:       return $\mathrm{Th}_H^m$
13:    end if
14:    {Variables not in $H$ are irrelevant}
15:    if $\mathbf{x}$ is a positive counterexample then
16:       $u := m$ and $P := \{v \in H : \mathbf{x}(v) = 1\}$
17:    else
18:       $\ell := m$ and $N := \{v \in H : \mathbf{x}(v) = 1\}$,
19:    end if
20: end while
21: $(P, \hat{v}) :=$ BinarySearch$(\emptyset, P)$
22: $\textsc{Base} := P \cap N$, $N' := N \setminus \textsc{Base}$, $P' := P \setminus \textsc{Base}$
23: while $|P'| > 1$ do
24:    $\mathrm{changed}H :=$ MakeEven$(\textsc{Base}, N', P', \hat{v}, H)$    {Uses at most 1 $\mathrm{MQ}$}
25:    if $\mathrm{changed}H$ then
26:       goto Line 1
27:    end if
28:    Let $N_0, N_1$ (resp. $P_0, P_1$) be an equal-sized partition of $N'$ (resp. $P'$)
29:    Ask $\mathrm{MQ}\left(\mathbf{1}_{\textsc{Base} \cup N_j \cup P_k}\right)$ for $j, k = 0, 1$
30:    Let $j$ and $k$ be indices s.t. $\mathrm{MQ}\left(\mathbf{1}_{\textsc{Base} \cup N_j \cup P_k}\right) = 0$ {such $j$ and $k$ exist}
31:    $\textsc{Base} := \textsc{Base} \cup P_k$, $P' := P_{1-k}$, $N' := N_j$
32: end while
33: $H := H \setminus N'$
34: goto Line 6
```

Algorithm 7 Function `MakeEven`$(\textsc{Base}, N', P', \hat{v}, H)$

```
1:  $\textsc{Test} := (\textsc{Base} \cup P') \setminus \{\hat{v}\}$
    {For any $v \in N'$, $\mathrm{MQ}\left(\mathbf{1}_{\textsc{Test}\cup\{v\}}\right) = 1$ iff $v$ is relevant}
2:  if $|P'|$ is odd then
3:      Choose $v_P \in P'$ arbitrarily and move $v_P$ from $P'$ to $\textsc{Base}$
4:      Choose $v_N \in N'$ arbitrarily and remove $v_N$ from $N'$
5:      if $\mathrm{MQ}\left(\mathbf{1}_{\textsc{Test}\cup\{v_N\}}\right) \neq 1$ then {$v_N$ irrelevant}
6:          $H := H \setminus \{v_N\}$
7:          return true {$H$ was modified}
8:      end if
9:  end if
10: if $|N'|$ is odd then
11:     Choose $v'_N \in N'$ arbitrarily and remove $v'_N$ from $N'$
12: end if
13: return false   {$H$ was not modified.}
```

search over threshold values in the first **while** loop (Lines 9–20), N is always assigned negative counterexamples from equivalence queries and P is always assigned positive counterexamples from equivalence queries.

Now we need to argue that at the end of that binary search (i.e., after Line 20) $|N| \geq |P|$ will hold. Consider the last time that N is updated. (This could be either when $\ell = 1$ before the **while** loop or inside the **while** loop.) After that update, N will consist of the variables from the negative counterexample that are not known to be irrelevant. That is, N is set to be $\{v \in H : \mathbf{x}_N(v) = 1\}$, where $\mathbf{x}_N$ was the counterexample from the equivalence query to Th_H^m (or to Th_H^1 if this was before the **while** loop). Since $\mathbf{x}_N$ was a *negative* counterexample it must be that $\mathrm{Th}_H^m(\mathbf{1}_N) = 1$. Thus it must be that $|N| \geq m$. In the control of the binary search over threshold values, the lower bound ℓ now becomes m, and ℓ is not updated again. Thus this value of ℓ is the value of ℓ after the loop has ended, and $|N| \geq \ell$ from now on.

Similar conditions hold for P and u, the upper bound in the control of the binary search. After the last update to P, it must be that $|P| < m$ (since P is a positive counterexample), u is updated to be this m, and u is not updated again. Thus $|P| < u$.

When the **while** loop terminates, $u \leq \ell + 1$. Since $|P| < u \leq \ell + 1$, it holds that $|P| \leq \ell$. Since $|N| \geq \ell$, we now have Equation (5.1).

Now we want to use N and P to construct three sets with what we call the "**key property:**"

Key property: *A triple of sets of variables* $(\textsc{Base}, N', P')$ *satisfies the key property for (target) threshold function* Th_θ^R *if the sets are pairwise disjoint, and it holds that*

- $\textsc{Base} \cup N'$ *is negative,*
- $|(\textsc{Base} \cup P') \cap R| = \theta$ *(i.e.,* $\textsc{Base} \cup P')$ *is a minimal positive set), and*
- $|N'| \geq |P'|$.

Given N and P satisfying Equation (5.1), in Line 21 P is set to be the set returned by `BinarySearch`$(\emptyset, P)$, which makes P a minimal positive set. We then set $\textsc{Base} = N \cap P$, and $P' = P \setminus \textsc{Base}$ and $N' = N \setminus \textsc{Base}$. The key property must hold for this triple: $N = \textsc{Base} \cup N'$ is negative; $P' = \textsc{Base}' \cup P$ is a minimal positive set, and it must be that $|N'| \geq |P'|$.

The following claim gives two important features of the key property.

Claim 5.4 (a) *If* $(\textsc{Base}, N', P')$ *satisfies the key property, then* N' *contains an irrelevant variable and* P' *contains a relevant variable.*
(b) *If* $(\textsc{Base}, N', P')$ *satisfies the key property and* $|P'| = 1$*, then every element of* N' *is irrelevant.*

The overall goal now is to find at least one of the irrelevant variables in N' and delete it. From now on the key property is maintained among the three sets, but in such a way that in each iteration the size of N' and P' gets halved. For this the algorithm splits up N' (respectively P') into two equal-sized disjoint subsets N_1 and N_1 (resp. P_0 and P_1). When both $|N'|$ and $|P'|$ are even then we can do this without any problem; otherwise we have to make some adjustments to N' and/or to P', that will be taken care of by procedure `MakeEven`, which we will describe presently.

Assume for now that both $|N'|$ and $|P'|$ are even. Let $\theta' = \theta - |R \cap \textsc{Base}|$. It holds that $|R \cap (N_0 \cup N_1)| < \theta'$ and $|R \cap (P_0 \cup P_1)| = \theta'$. Thus for some $j, k \in \{0, 1\}$ we have $|R \cap (N_j \cup P_k)| < \theta'$ (equivalently $\mathrm{MQ}(\mathbf{1}_{\textsc{Base} \cup N_j \cup P_k}) = 0$). Note that the sets $\textsc{Base} := \textsc{Base} \cup P_k$, $N' := N_j$ and $P' := P_{1-k}$ still have the key property, but the size of N' and P' is reduced by half. Thus after at most $\log n$ steps P' is reduced to a set consisting of a single (relevant) variable. Thus N' is a nonempty set of irrelevant variables (part (b) of Claim 5.4) that can be removed from H (Line 33).

Finally, the function `MakeEven`$(\textsc{Base}, N', P', \hat{v}, H)$ works as follows. Its job is to move variables among sets so as to preserve the key property for $\textsc{Base}$, N', and P', while making both N' and P' have even size. Sometimes instead, however, it will remove an irrelevant variable from H—in this case it returns *true* and its caller restarts with the smaller H.

First `MakeEven` checks whether $|P'|$ is odd, and if so, it moves an arbitrary element v_P of P' to $\textsc{Base}$. Note that if v_P was relevant, this action might turn $\textsc{Base} \cup N'$ into a positive set; thus the key property might be violated; so an arbitrary element v_N will also be removed from N'. If v_N is irrelevant (which can be tested using set $\textsc{Test}$ defined at Line 1), `MakeEven` removes it from H and immediately returns true, so the overall search can be restarted.

Otherwise (i.e, if v_N is relevant, or if `MakeEven` was called with P' of even cardinality) the key property holds for the new triple $(\textsc{Base}, N', P')$, and $|P'|$ is even. Then `MakeEven` checks if $|N'|$ is odd, and if so, an arbitrary v'_N gets removed from N'.

If `MakeEven` returns false (no irrelevant variable was removed from H), then the resulting triple will also have the key property.

Now we give the complexity analysis.

For each deletion found, at most $2 + \lceil \log_2 n \rceil$ equivalence queries are used to get the sets N and P, and then one call to `BinarySearch` to make P a minimal positive set. Next the algorithm iterates, shrinking both $|P'|$ and $|N'|$ by half in each iteration, at most $\lceil \log_2 n \rceil$ times. Each such iteration requires at most 5 membership queries. Thus (as `BinarySearch` always uses at most $\lceil \log_2 n \rceil$ membership queries per call) the deletions require at most $O(|H \setminus R| \log n)$ queries. □

Now we can state the main result of the section.

Theorem 5.5 *Let the φ be the initial and ψ the target formula, where both are n-variable threshold funtions. Then* `ReviseThreshold`(φ)*, using $O(\hat{e} \log n)$ queries, outputs ψ, where $\hat{e} = \mathrm{dist}(\varphi, \psi)$.*

Proof
First, two membership queries are used to determine if the target is either of the two constant Boolean functions. For nonconstant functions, the complexity and the correctness follow from Lemmas 5.2 and 5.3. □

5.3 Example Run of `ReviseThreshold`

To demonstrate the algorithm, we provide an example run.

Let $\mathcal{V}_8$ be the set of variables in focus, furthermore let the initial function φ and the unknown target function ψ be

$$\begin{aligned} \varphi &= \mathrm{Th}^1_{\{v_1,v_2,v_4\}} \\ \psi &= \mathrm{Th}^4_{\{v_1,v_2,v_3,v_5,v_6\}} . \end{aligned}$$

First, in subsection 5.3.1 we determine all the relevant variables that were left out from $\{v_1, v_2, v_4\}$, then in subsection 5.3.2 we further revise our hypotheses from subsection 5.3.1 by removing those irrelevant variables that appeared in $\{v_1, v_2, v_4\}$.

5.3.1 Adding the Previously Unknown Relevant Variables

Two MQ's to 00000000 and 11111111 determine that the target function is nonconstant.

We next determine the necessary additions, that is, the relevant variables from $\{v_3, v_5, v_6, v_7, v_8\}$, using Procedure `FindAdditions`. Since assignment $\mathbf{1}_{\{v_1,v_2,v_4\}}$ is negative, $\textsc{Potentials} = \{v_3, v_5, v_6, v_7, v_8\}$ must contain some unknown relevant variables.

In Lines 12–19 of Procedure `FindAdditions`, we repeatedly use `BinarySearch` from $\textsc{Base} = \{v_1, v_2, v_4\}$ to $\textsc{Base} \cup \textsc{Potentials}$ to find one. Inside `BinarySearch` ask $\mathrm{MQ}(11111100)$, the answer is 1. Ask $\mathrm{MQ}(11111000)$, the answer is 1. Ask $\mathrm{MQ}(11110000)$, the answer is 0. The last negative and positive examples differ by the single variable v_5—thus v_5 is relevant, and is returned to `FindAdditions`, and `FindAdditions` adds v_5 to $\textsc{NewRelevants}$.

Now exclude the newly found relevant variable v_5 from consideration. As $\mathbf{1}_{\textsc{Base}\cup\{v_3,v_6,v_7,v_8\}}$ is still positive, we make another similar call to `BinarySearch`. Ask $\mathrm{MQ}(11110100)$, the answer is 1. Ask $\mathrm{MQ}(11110000)$, the answer is 0. The last positive and negative vectors differ only on v_6 —thus v_6 is relevant, and is added to $\textsc{NewRelevants}$. Excluding v_6 from consideration too, we find that $\mathbf{1}_{\textsc{Base}\cup\{v_3,v_7,v_8\}}$ is negative. This means that the number of relevant variables in $\{v_1, v_2, v_4\} \cup \{v_3, v_6, v_7, v_8\}$ is the same as the unknown threshold. So, we update $\textsc{Base}$ from $\{v_1, v_2, v_4\}$ to $\{v_1, v_2, v_4, v_6\}$, and do `BinarySearch` from $\textsc{Base}$ to $\textsc{Base} \cup \{v_3, v_7, v_8\}$. Ask $\mathrm{MQ}(11110110)$, the answer is 1. Ask $\mathrm{MQ}(11110100)$, the answer 1. Ask $\mathrm{MQ}(11010100)$, the answer is 0—thus v_3 is relevant. Testing $\mathbf{1}_{\{v_1,v_2,v_3,v_4,v_6\}}$, we find that it is positive; thus since the number of relevant variables in $\{v_1, v_2, v_3, v_4, v_6, v_7, v_8\}$ is the same as the threshold, we know that $\{v_7, v_8\}$ contains no relevant variables.

5.3.2 Deleting the Irrelevant Variables

Now we know that $H = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ contains all the relevant variables; all that left is to get rid of the irrelevant ones (and determine the threshold).

This is done in `FindDeletions`. Procedure `FindDeletions` first determines a "big" positive and a "small" negative set. Suppose that we ask equivalence queries for Th_H^θ, for $\theta = 1, \ldots, |H|$. Since ψ is not constant, we must find two θ-values ℓ and u, and corresponding counterexamples $\mathbf{1}_P$ and $\mathbf{1}_N$, such that $u = \ell + 1$, P is positive, and N is negative. Then it must also hold that $|P| \leq u - 1 = \ell \leq |N|$; thus N must contain an irrelevant element. In fact, we determine the above ℓ, u, P and N using binary search on the threshold value θ.

First, in Lines 1–6 we ask the two extreme cases $\mathrm{EQ}\left(\mathrm{Th}_H^{|H|}\right)$ and $\mathrm{EQ}\left(\mathrm{Th}_H^1\right)$, getting counterexamples, say, 111110 and 000111 [1]. The remainder of this binary search over threshold values is carried out in Lines 9–20. Ask $\mathrm{EQ}\left(\mathrm{Th}_H^4\right)$, and suppose we receive the negative counterexample 001111. Ask $\mathrm{EQ}\left(\mathrm{Th}_U^5\right)$, and suppose we receive the positive counterexample 111010. Now we have $u = 5$, $\ell = 4$, $P = \{v_1, v_2, v_3, v_5\}$ and $N = \{v_3, v_4, v_5, v_6\}$. Because P is already a minimal positive set, it does not change in the call to `BinarySearch` at Line 21.

Now, with the help of P, we determine an irrelevant variable of N as follows. We set their common part to be $\textsc{Base} = \{v_3, v_5\}$. The remaining parts of P and N, which are $P' = \{v_1, v_2\}$ and $N' = \{v_4, v_6\}$ are both even, so the call to `MakeEven` makes no changes (and returns *false*). We cut this remaining part of P' (resp. N') in two equal parts: $P_1 = \{v_1\}$ and $P_2 = \{v_2\}$ (resp. $N_1 = \{v_4\}$ and $N_2 = \{v_6\}$). Asking membership queries for all combinations $\textsc{Base} \cup P_i \cup N_j$, $i, j = 1, 2$, we find that $\textsc{Base} \cup P_1 \cup N_1$ is negative, meanwhile $\textsc{Base} \cup P_1 \cup P_2$ is positive. As P_2 has

[1] As v_7 and v_8 are known to be irrelevant, from here on we shall omit the corresponding bits in the examples.

cardinality 1, this means that v_4 is irrelevant; remove it from H.

Now we restart, and conduct a binary search on the threshold value again, with the difference, that now $H = \{v_1, v_2, v_3, v_5, v_6\}$. Ask $\mathrm{EQ}\left(\mathrm{Th}_H^3\right)$, and suppose we receive the negative counterexample 111000. Then asking $\mathrm{EQ}\left(\mathrm{Th}_H^4\right)$ we receive (), meaning that the learning process has come to a successful end.

5.4 Lower Bounds on Revising Threshold Formulas

In this section, we show that both types of queries are needed for the efficient revision of threshold functions, and that the query complexity of our algorithm is essentially optimal up to order of magnitude. The first result shows that efficient revision is not possible with membership queries alone, even if we allow a restricted type of equivalence queries as well, and the second result shows that efficient revision is not possible with equivalence queries alone.

Theorem 5.6 *Assume that both the initial formula and the target formula have threshold value t, and that the learner is allowed to ask equivalence queries only for threshold functions also having threshold value t. (On the other hand, no restrictions are set on the membership queries.) Under this restriction, the query complexity of revising threshold formulas is at least $n - 1$ (where n is the number of variables in the universe in scope), even when the revision distance is only one.*

Proof
Let the initial function be $\mathrm{Th}_{\mathcal{V}_n}^{n-1}$, let $\psi_i := \mathrm{Th}_{\mathcal{V}_n \setminus \{v_i\}}^{n-1}$ for $1 \le i \le n$, and set $\Psi := \{\psi_i : 1 \le i \le n\}$.

Consider the following scenario. When the learner asks a membership query $\mathrm{MQ}(\mathbf{1}_V)$ for some $V \subseteq \mathcal{V}_n$, then the answer is

- 0, if $|V| < n - 1$. In this case Ψ remains unchanged.
- 1, if $|V| = n$ or if $\Psi = \{\mathrm{Th}_V^{n-1}\}$. Again, Ψ remains unchanged.
- 1, if $V = \mathcal{V}_n \setminus \{v_i\}$ and $\Psi = \{\psi_i\}$.
- 0 otherwise. Also, remove ψ_i from Ψ for i with $\{v_i\} = \mathcal{V}_n \setminus V$.

When the learner asks an equivalence query $\mathrm{EQ}(\mathrm{Th}_U^{n-1})$ for some $U \subseteq \mathcal{V}_n$, then the assignment returned is $\mathbf{x}$, where

- if $|U| < n - 1$ (i.e., the hypothesis is constant 0), then $\mathbf{x}$ is the positive counterexample $\mathbf{1}$. In this case Ψ remains unchanged.
- if $|U| = n$ then $\mathbf{x}$ is the negative counterexample $\mathbf{1}^{(v_i \mapsto 0)}$ for some i satisfying $|\Psi \setminus \{\psi_i\}| \ge 1$. Also, remove ψ_i from Ψ.
- if $\Psi = \{\mathrm{Th}_U^{n-1}\}$, then $\mathbf{x} = ()$.

- otherwise $\mathbf{x}$ is the negative counterexample $\mathbf{1}_U$. Also, remove ψ_i from Ψ for i with $\{v_i\} = \mathcal{V}_n \setminus V$.

Note that during the whole process each element of the actual Ψ is consistent with all the previous informations, and that after each query $|\Psi|$ decreases by at most one. But, as the learning process cannot end as long as there are more than one non-equivalent hypotheses consistent with the previous informations, it follows that the learner must ask at least $n-1$ queries. $\square$

Theorem 5.7 *The query complexity of revising threshold formulas with equivalence queries alone is at least $n-1$ (where n is the number of variables in the universe in scope), even when the revision distance is only one.*

Proof
Set $n = 2k$, and let the initial function $\mathrm{Th}^k_{\mathcal{V}_n}$. Also, for $k+1 \le i \le n$, let $\psi_i := \mathrm{Th}^k_{\mathcal{V}_n \setminus \{v_i\}}$, and set $\Psi := \{\psi_i : k+1 \le i \le n\}$

Consider the following scenario. When the learner asks an equivalence query $\mathrm{EQ}\left(\mathrm{Th}^\ell_U\right)$ for some $U \subseteq \mathcal{V}_n$, then the assignment returned is $\mathbf{x}$, where

- if $\ell < k$ and
  - $|U| \ge \ell$, then $\mathbf{x}$ is the negative counterexample $\mathbf{1}_{U'}$, where U' is an arbitrary subset of U with cardinality ℓ. In this case Ψ remains unchanged.
  - otherwise (i.e., if Th^ℓ_U is constant 0), then $\mathbf{x}$ is the positive counterexample $\mathbf{0}^{(v_1 \mapsto 1, \ldots, v_k \mapsto 1)}$. Again, Ψ remains unchanged.
- if $\ell > k$, then $\mathbf{x}$ is the positive counterexample $\mathbf{0}^{(v_1 \mapsto 1, \ldots, v_k \mapsto 1)}$. Again, Ψ remains unchanged.
- if $\ell = k$ and
  - if $U \supseteq \{v_{k+1}, \ldots, v_n\}$, then $\mathbf{x}$ is the negative counterexample $\mathbf{1}^{(v_1 \mapsto 0, \ldots, v_k \mapsto 0)}$. Again, Ψ remains unchanged.
  - otherwise, if $\{v_1, \ldots, v_k\} \not\subseteq U$, then $\mathbf{x}$ is the positive counterexample $\mathbf{0}^{(v_1 \mapsto 1, \ldots, v_k \mapsto 1)}$. Again, Ψ remains unchanged.
  - if $\Psi = \left\{\mathrm{Th}^\ell_U\right\}$, then $\mathbf{x} = ()$.
  - otherwise $\mathbf{x}$ is the positive counterexample $\mathbf{1}_{\{2,\ldots,k\} \cup \{i\}}$ for some i with $v_i \in \{v_{k+1}, \ldots, v_n\} \setminus U$ (note that it must be the case that U contains all of $v_1, \ldots, v_k$, and is missing at least one of $v_{k+1}, \ldots, v_n$). Also, remove ψ_i from Ψ.

Note that during the whole process each element of the actual Ψ is consistent with all the previous informations, and that after each query $|\Psi|$ decreases by at most one. But, as the learning process cannot end as long as there are more than one non-equivalent hypotheses consistent with the previous informations, it follows that the learner must ask at least $n-1$ queries. $\square$

Now we show that the query bound of algorithm `ReviseThreshold` cannot be improved for small values of $\hat{e}$ (i.e., constant $\hat{e}$), and cannot be much improved in general. We gave a revision algorithm with query complexity $O(\hat{e}\log n)$; we give here the close lower bound of $\Omega(\hat{e}\log(n/\hat{e}))$. (We think that the first one is closer to the real answer)

Proposition 5.8 *The query complexity of revising threshold formulas with membership and equivalence queries is $\Omega(\hat{e}\log(n/\hat{e}))$, where n is the number of variables in the universe in scope and $\hat{e}$ is the revision distance between the initial formula and the target formula.*

Proof
Put $\varphi = \mathrm{Th}^1_\emptyset$, and let $\mathcal{R} = \left\{\mathrm{Th}^{\hat{e}}_R : R \subseteq \mathcal{V}_n, |R| \le \hat{e}\right\}$. Now each element of $\mathcal{R}$ is equivalent to some clause of size at most $\hat{e}$ over $\mathcal{V}_n$. As the class of these clauses has Vapnik-Chervonenkis dimension $\Omega(\hat{e}\log(n/\hat{e}))$ [92], the claimed bound for the query complexity follows (see Section 3.3).

□

The following result answers the question that arises naturally whenever one is learning threshold functions: why not use `Winnow` [2]? After all it is one of the most successful tools for learning threshold functions. Furthermore, it can be successfully used for revision in some cases (see, e.g. Chapter 6). The answer is simple and somewhat surprising: under our settings, using `Winnow` as defined in [92] would result in an *inefficient* revision algorithm.

Proposition 5.9 `Winnow` *is not an efficient revision algorithm for threshold functions. More precisely, for any weight vector representing the initial threshold function $\mathrm{Th}^1_{v_1,\ldots,v_n}$,* `Winnow` *can make n mistakes when the target function is $\mathrm{Th}^2_{v_1,\ldots,v_n}$.*

Proof
The statement follows easily, noting that the weight of each relevant variable is at least as big as the threshold used by `Winnow`, thus giving `Winnow` the negative examples $\mathbf{1}_{\{v_1\}}, \ldots, \mathbf{1}_{\{v_n\}}$ one after another, it will evaluate to 1 for each of them. □

5.5 Concluding Remarks

It would be interesting to consider disjunctions of a bounded number of threshold functions in the revision model. This class is a generalization of monotone DNF with a bounded number of terms, which can be revised efficiently [53]. It is also related to the robust logic framework of Valiant [128] mentioned in the introduction.

Finally note that the results presented in this chapter—unless noted otherwise—appeared in the paper [116], co-authored by the author of the present dissertation.

[2]See Chapter 6 for more on `Winnow`.

Chapter 6

Projective DNF Formulas

The notion of *projection learning* was introduced by Valiant [128], motivated by constraints imposed on learnability by biology. Projection learning aims to learn a target concept over some large domain (in our case $\mathcal{A}_n$), by learning some of its projections—or rather: restrictions—to a class of smaller domains, and combining these projections. Valiant proved a general mistake bound for the resulting algorithm under certain conditions. The basic assumption underlying projection learning is that there is a family of simple projections that cover all positive instances of the target, where simple means belonging to some efficiently learnable class. The projections describing the target in this way can also be thought of as a set of experts, each specialized to classify a subset of the instances, such that whenever two experts overlap they always agree in their classification.

Perhaps the most natural special case of this framework, also discussed by Valiant, is when the projection domains are subcubes of a fixed dimension, and the restrictions of the target to these domains are conjunctions. In this case, the algorithm learns a class of disjunctive normal forms (DNF) called projective DNF (precise definitions will be given later). The class of projective DNF expressions does not appear to have been studied at all before Valiant's work. As the learnability of DNF is shown to be a hard problem in computational learning theory [1], it is of interest to those who study computational learning theory to identify new learnable subclasses and to understand their scope.

In this chapter an efficient revision algorithm is presented for the class of projective DNFs in the *mistake bounded* model for the *general case*. Additionally some (learnability related) combinatorial properties of this class is annalyzed. More precisely lower and upper bounds for the *exclusion dimension* of projective DNF. The exclusion dimension, or certificate size [11; 65; 67], of a formula class is closely related to its learning complexity in the model of proper learning with equivalence and membership queries. This way bounds are obtained for the complexity of learning projective DNF in this model as well.

Finally, note that this chapter does not contain an example run—contrary to the

[1]Alekhnovich et al. showed that DNF is not properly PAC learnable in polynomial time unless NP = RP [5], providing further motivation to find positive learnability results.

two previous ones dealing also with results on revision. The main reason for this is that the algorithm itself is much more simple than the ones presented in the two previous chapters (however this does not seem to hold for the *analysis* of the algorithm), and thus an example run would not provide further insights about the algorithm.

6.1 Further Definitions and Notations

First we introduce projective disjunctive normal forms and we briefly discuss some of their properties.

Definition 6.1 *A DNF formula φ is a k-***projective DNF***, or k-***PDNF** *if it is of the form*

$$\varphi = \rho_1 t_1 \vee \cdots \vee \rho_\ell t_\ell, \tag{6.1}$$

*where, for $i = 1, \ldots, \ell$, ρ_i is a k-conjunction (called the ρ-***part** *of the term $\rho_i t_i$), t_i is a conjunction (called the t-***part** *of the term $\rho_i t_i$) and it holds that*

$$\rho_i \varphi \equiv \rho_i t_i. \tag{6.2}$$

*A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is k-***projective** *if it can be written as a k-PDNF formula. The class of n-variable k-projective functions is denoted by k-PDNF$_n$.*

The k-conjunctions ρ_i are also called k-**projections**, or, when k is clear from context, simply **projections**. Conditions (6.1) and (6.2) mean that when restricted to the subcube $\mathcal{T}(\rho_i)$, the formula φ is equivalent to the conjunction t_i, and every true point of φ arises this way for some restriction. This corresponds to the intuition, described earlier, that the restrictions to a prespecified set of simple domains are simple, and the whole function can be patched together from these restrictions.

Note that in order to specify a k-PDNF, it is *not* sufficient to specify its terms, but for each term one has to specify its ρ-part and its t-part; that is, the projection and the corresponding conjunction have to be distinguished. If necessary, we indicate this distinction by placing a dot between the two parts. For example,

$$(x \cdot y) \vee (z \cdot y) \quad \text{and} \quad (x \cdot y) \vee (\overline{x} \cdot yz) \tag{6.3}$$

are two different 1-PDNF for the same function. The dots are omitted whenever this does not lead to confusion. The conjunctions ρ_i and t_i may have common literals. The requirement (6.2) is equivalent to requiring that

$$\rho_j \rho_i t_i \equiv \rho_i \rho_j t_j \tag{6.4}$$

for every i and j. This makes it easy to verify that a given expression, such as those in (6.3), is indeed a k-PDNF. It also shows that the disjunction of any set of terms of a k-PDNF is again a k-PDNF.

If a function is k-projective, then it is k'-projective for every k' with $k \leq k' \leq n$. Note that the complete DNF (consisting of n-conjunctions corresponding to the true points of f) shows that every n-variable function is n-projective.

For more on projective DNFs and their relations with some other basic formula classes (like k-DNFs, k-term-DNFs and decision lists) see [115].

6.1.1 Revision

In addition to the standard mistake-bounded model, as a technical tool for the learning result, we also consider a model of learning in the presence of noise. In the model of learning monotone disjunctions with **attribute errors** (Auer and Warmuth [18], also used by Valiant [128] with a different name) it may happen that y is *not* the correct classification of $\mathbf{x}$, that is, $f_{\text{trg}}(\mathbf{x}) \neq y$. It is assumed that the error comes from some components (or attributes) of $\mathbf{x}$ being incorrect, and the number of attribute errors committed in a round is the minimal number of components that need to be changed in order to get the correct classification. More precisely, if in round r the classification y_r is not the correct classification of $\mathbf{x}_r$, then, if $y_r = 1$ then $\textsc{AttrErr}(r) = 1$ (as it is enough to switch one bit on to satisfy a disjunction), and if $y_r = 0$ then $\textsc{AttrErr}(r)$ is the number of variables that are included in the target disjunction and which are set to 1 in $\mathbf{x}_r$. The total number of attribute errors for a given run, denoted $\#\textsc{AttributeErrors}$, is the sum of the attribute errors of the rounds. This notion is used only for technical purposes: it plays an important role inside some proof, but does not appear in any results.

The **revision operations** are the deletion of a literal or a term, the addition of a new empty term of the form $\rho \cdot \top$, and the addition of a literal.

The **revision distance** of two terms t and t^* is the number of literals occurring in exactly one of the two terms, denoted $|t \triangle t^*|$. Similarly, the distance between two disjunctions is also the number of literals occurring in exactly one of the two disjunctions.

The **revision distance** between an *initial* k-PDNF formula φ and a *target* k-PDNF formula ψ of the form

$$\begin{aligned} \varphi &= \rho_1 t_1 \vee \cdots \vee \rho_\ell t_\ell \vee \rho_{\ell+1} t_{\ell+1} \vee \cdots \vee \rho_{\ell+s} t_{\ell+s}, \\ \psi &= \rho_1 t_1^* \vee \cdots \vee \rho_\ell t_\ell^* \vee \rho_1' t_1' \vee \cdots \vee \rho_a' t_a' \end{aligned}$$

is

$$\operatorname{dist}(\varphi, \psi) = s + \sum_{i=1}^{\ell} |t_i \triangle t_i^*| + \sum_{i=1}^{a} (|t_i'| + 1),$$

where $\{\rho_{\ell+1}, \ldots, \rho_{\ell+s}\} \cap \{\rho_1', \ldots, \rho_a'\} = \emptyset$. For example, the s term in the definition of $\operatorname{dist}(\varphi, \psi)$ corresponds to the deletion of the s terms $\rho_{\ell+1} t_{\ell+1}, \cdots, \rho_{\ell+s} t_{\ell+s}$.

Given an initial formula φ and a target formula ψ, we want our mistake bound to be polynomial in the revision distance $\hat{e} = \operatorname{dist}(\varphi, \psi)$, and logarithmic (or polylogarithmic) in all other parameters. In this case, that means logarithmic in n and, for k-PDNF, in the total number of projections of size k, which is $2^k \binom{n}{k}$.

Note that this is in accordance with the general approach described in Chapter 3.

6.2 Revision Algorithm for Disjunctions and for k-PDNF Formulas

The main tool in Valiant's learning algorithm for projective DNFs [128] is Littlestone's `Winnow` algorithm [92], which is a kind of multiplicative version of the well-known `Perceptron` algorithm. We begin by demonstrating that the original `Winnow` with appropriately modified initial weights is an efficient revision algorithm in the mistake bounded model for disjunctions, even in the presence of attribute errors—if we are willing to tolerate a number of mistakes polynomial in the number of attribute errors as well as the usual parameters. We will use this result to show how to use an algorithm similar to Valiant's PDNF learning algorithm to revise PDNF. The overall algorithm has a two-level structure, with many instances of a revision version of Winnow on the lower level feeding their outputs to one instance of a revision version of Winnow on the top level. Note that, even with noise-free data, mistakes made by the lower-level `Winnows` will represent attribute errors in the input to the top-level `Winnow`.

6.2.1 Revising Disjunctions

Algorithm `RevWinn` (pseudocode displayed as Algorithm 8) revises a monotone disjunction. It can be applied to revise an arbitrary disjunction by introducing extra variables for the negated literals, and this in turn can be used to revise arbitrary conjunctions by applying the De Morgan rules. We now present `RevWinn`; we will later assume without further discussion that we have versions available for arbitrary disjunctions and for conjunctions

Let the set of variables in focus be some finite $V \subseteq \mathcal{V}$. Algorithm `RevWinn` revises an initial disjunction φ over V. It maintains a weight vector $\mathbf{w}$ of length $|V|$, which determines the current hypothesis, and is updated each time a mistake is made. We use $\mathbf{w}_r$ to denote its value *after* round r. Accordingly $\mathbf{w}_0$ denotes the initial weight vector [2].

The algorithm consists of three main parts: initialization of the weight vector $\mathbf{w}$ (which initializes the hypothesis), prediction (the hypothesis part), and the update part. Formally, we break out each as a subroutine to make later discussion easier.

Let us now describe these three parts of `RevWinn`. The initialization part is done by using function `Init`, which, on input V and φ outputs a vector $\mathbf{w}$ of length V (and

[2]Actually, this is Littlestone's `Winnow2` [92] using different initial weights—with his parameters set to $\alpha = 2$, and $\theta = |V|/2$—, except that the weights are all devided by $|V|$, because this seems to make the analysis a little easier to follow.

Algorithm 8 Algorithm $\texttt{RevWinn}(V, \varphi)$

```
1: $\mathbf{w} := \texttt{Init}(V, \varphi)$ {initialize the weight vector}
2: for round $r = 1, 2, \ldots$ do
3:    {The input in round $r$ is the instance $\mathbf{x}_r$ with domain $V$}
4:    Output prediction $\hat{y}_r := \texttt{h}(\mathbf{x}_r, \mathbf{w})$
5:    if receiving label $y_r$ for $\mathbf{x}_r$ it holds that $\hat{y}_r \neq y_r$ then
6:       {the algorithm made a mistake, so update the weights}
7:       $\mathbf{w} := \texttt{Update}(y_r, \mathbf{x}_r, \mathbf{w})$
8:    end if
9: end for
```

indexed by the variables in V), with

$$\mathbf{w}(v) = \begin{cases} 1 & \text{if variable } v \text{ appears in } \varphi, \\ 1/|V| & \text{otherwise,} \end{cases}$$

for $v \in V$.

Given weight vector is $\mathbf{w}$, the hypothesis function evaluates

$$\texttt{h}(\mathbf{x}, \mathbf{w}) = \begin{cases} 0 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle \text{ is less than } 1/2, \\ 1 & \text{otherwise} \end{cases}$$

on input instance $\mathbf{x}$ (with domain V), where

$$\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{v \in V} \mathbf{w}(v) \cdot \mathbf{x}(v)$$

is the **dot product** of $\mathbf{w}$ and $\mathbf{x}$. The hypothesis is used to make predictions; in round r the algorithm predicts that the label of $\mathbf{x}_r$ is $\hat{y}_r = \texttt{h}(\mathbf{x}_r, \mathbf{w}_{r-1})$.

Finally the function $\texttt{Update}(y, \mathbf{x}, \mathbf{w})$, returns a vector $\mathbf{w}'$, a modification of the weight vector $\mathbf{w}$:

$$\mathbf{w}'(v) = \mathbf{w}(v) \cdot 2^{(y-\hat{y})\cdot \mathbf{x}(v)} = \begin{cases} 2 \cdot \mathbf{w}(v) & \text{if } y > \hat{y} \text{ and } \mathbf{x}(v) = 1, \\ (1/2) \cdot \mathbf{w}(v) & \text{if } y < \hat{y} \text{ and } \mathbf{x}(v) = 1, \\ \mathbf{w}(v) & \text{otherwise,} \end{cases}$$

for $v \in V$, where $\hat{y}$ is the output of the hypothesis function on $\mathbf{x}$ (i.e., $\hat{y} = \texttt{h}(\mathbf{x}, \mathbf{w})$). This function does nothing and need not even be called if there is no mistake; that is, if $\hat{y} = y$.

Note that throughout, all of the weights are always in the interval $(0, 1]$. This can be seen using an induction argument as follows. Initially the statement is true. Now assume that the weights after round $r - 1$ are all between 0 and 1. If $y_r = \hat{y}_r$, then the weights are not changed. If $y_r = 0$ and $\hat{y}_r = 1$, then some weights are halved, and some unchanged—thus the statement will be true after round r. If $y_r = 1$ and $\hat{y}_r = 0$, then $\langle \mathbf{w}_{r-1}, \mathbf{x}_r \rangle$ is less then $1/2$, so the sum of the weights of components having 1 in assignment $\mathbf{x}_r$ is less then 1/2. As `RevWinn` doubles the weights of exactly these components, the statement will remain true after round r.

Theorem 6.2 *The number of mistakes made by Algorithm* `RevWinn` *with initial (monotone) disjunction φ and target (monotone) disjunction ψ is*

$$O(\#\text{AttributeErrors} + \hat{e}\log n),$$

where $\hat{e} = \operatorname{dist}(\varphi, \psi)$, $n = |V|$ and V is the set of variables in focus.

Proof
Consider any run of the algorithm of length R. Let I be the set of variables $v \in V$ that appear in both the initial and target disjunctions, such that for at least one round r variable $\mathbf{x}_r(v) = 1$ but $y_r = 0$. Let $J \subseteq V$ be the set of variables that appear in the target disjunction but not in the initial disjunction. Let us also introduce the notation $\overline{I \cup J} = V \setminus (I \cup J)$.

We will use later the fact that any variable in both φ and ψ that is *not* in I never has its weight changed from 1.

For the proof we use a potential function $\Phi(\mathbf{w})$ that is somewhat different from those used in some other cases for analyzing `Winnow` (e.g., in [18; 80]). Put $\Phi(\mathbf{w}) = \sum_{v\in V}^{n} \Phi_v(\mathbf{w})$, where

$$\Phi_v(\mathbf{w}) = \begin{cases} \mathbf{w}(v) - 1 + \ln(1/\mathbf{w}(v)) & \text{if } v \in I \cup J, \\ \mathbf{w}(v) & \text{otherwise.} \end{cases}$$

It can be verified that $\Phi_i(\mathbf{w}) \geq 0$ for any $\mathbf{w} \in (0,1]^n$.

Let $\Delta_r = \Phi(\mathbf{w}_{r-1}) - \Phi(\mathbf{w}_r)$ denote the change of the potential function during round r. We will derive both upper and lower bounds on $\sum_{r=1}^{R} \Delta_r$ that will allow us to relate the number of mistakes made by `RevWinn` to $\hat{e}$, n, and $\#\text{AttributeErrors}$.

First we derive an upper bound:

$$\begin{aligned} \sum_{r=1}^{R} \Delta_r &= \Phi(\mathbf{w}_0) - \Phi(\mathbf{w}_R) \\ &\leq \Phi(\mathbf{w}_0) - \sum_{v\in\overline{I\cup J}} \mathbf{w}_R(v) \\ &= \sum_{i\in I} \Phi_i(\mathbf{w}_0) + \sum_{j\in J} \Phi_j(\mathbf{w}_0) + \sum_{v\in\overline{I\cup J}} (\mathbf{w}_0(v) - \mathbf{w}_R(v)). \end{aligned} \tag{6.5}$$

For $v \in I$ we initialized $\mathbf{w}_0(v) = 1$ so $\Phi_v(\mathbf{w}_0) = 0$. Also, $|J| \leq \hat{e}$, and $\Phi_v(\mathbf{w}_0) = \ln(2n) - (2n-1)/2n < \ln(2n)$ for $v \in J$, so the sum of the first two terms is at most $\hat{e}\ln(2n)$. Now we need to bound the third term. The variables that appear neither in ψ nor in φ have initial weights $1/(2n)$, and so altogether can contribute at most 1/2 to the sum. There are at most $\hat{e}$ variables in φ that are not present in ψ, so those variables can contribute at most $\hat{e}$ to the sum. Finally, as noted earlier, the weights never change for those variables in both φ and ψ but not in I. Thus we get

$$\sum_{r=1}^{R} \Delta_r \leq \hat{e} \ln 2n + \hat{e} + 1/2. \tag{6.6}$$

To get a lower bound on the sum, we begin by deriving a lower bound on the change in potential in one round. Now

$$\begin{aligned} \Delta_r &= \sum_{v \in I \cup J} \left(\mathbf{w}_{r-1}(v) - \mathbf{w}_r(v) + \ln \frac{\mathbf{w}_r(v)}{\mathbf{w}_{r-1}(v)} \right) + \sum_{v \in \overline{I \cup J}} (\mathbf{w}_{r-1}(v) - \mathbf{w}_r(v)) \\ &= \sum_{v \in V} (\mathbf{w}_{r-1}(v) - \mathbf{w}_r(v)) + \sum_{v \in I \cup J} \ln \frac{\mathbf{w}_r(v)}{\mathbf{w}_{r-1}(v)} \,. \end{aligned} \tag{6.7}$$

Examining the `RevWinn` code, one can see that there are three cases for updating weights at the end of a round r: no change in any weights, some or all weights are decreased—called a **demotion** round—, and some or all weights are increased—called a **promotion** round. Obviously, when no update is done in round r (i.e., $\hat{y}_r = y_r$), then $\Delta_r = 0$.

In a demotion round, $\hat{y}_r = 1$ and $y_r = 0$. By the definition of I and J, in this case $\textsc{AttrErr}(r) = |(I \cup J) \cap \{v : \mathbf{x}_r(v) = 1\}|$. Also, the total weight of components being on in $\mathbf{x}_r$ is at least 1/2 (recall how $\hat{y}_r$ is evaluated), and the weight of each of those components is halved. So, using (6.7),

$$\Delta_r \geq \frac{1}{4} + |(I \cup J) \cap \{v : \mathbf{x}_r(v) = 1\}| \ln \frac{1}{2} = \frac{1}{4} - (\ln 2)\textsc{AttrErr}(r)\,. \tag{6.8}$$

In a promotion round, $\hat{y}_r = 0$ and $y_r = 1$. We know that the components of $\mathbf{x}_r$ that are on have total weight less than $1/2$ (again, by the evaluation rule of $\hat{y}_r$), and that each of these components is multiplied by 2. So the first term in (6.7) is at least $-1/2$. Thus $\Delta_r \geq -1/2 + |(I \cup J) \cap \{v : \mathbf{x}_r(v) = 1\}| \cdot \ln 2$. Now if $y_r = \psi(\mathbf{x}_r)$, then $|(I \cup J) \cap \{v : \mathbf{x}_r(v) = 1\}| \geq 1$, because we know that $\hat{y}_r = 0$ and we know that all the weights of variables in both φ and ψ but not in I are 1. If $y_r \neq \psi(\mathbf{x}_r)$, then $\textsc{AttrErr}(r) = 1$. Thus, in a promotion round, it always holds that

$$\Delta_r \geq -1/2 + (\ln 2)(1 - \textsc{AttrErr}(r)). \tag{6.9}$$

Finally, let M^- denote the total number of demotions and M^+ the total number of promotions. Then (6.8) and (6.9) give us

$$\begin{aligned} \sum_{r=1}^{R} \Delta_r \;\geq\; & \sum_{\{r : \hat{y}_r = 1, y_r = 0\}} \left(\frac{1}{4} - (\ln 2)\textsc{AttrErr}(r) \right) \\ & + \sum_{\{r : \hat{y}_r = 0, y_r = 1\}} \left(\ln 2 - \frac{1}{2} - (\ln 2)\textsc{AttrErr}(r) \right) \end{aligned}$$

$$= \frac{M^-}{4} + \left(\ln 2 - \frac{1}{2}\right) M^+ - (\ln 2)\#\textsc{AttributeErrors}.$$

Combining this with (6.6) gives the desired mistake bound. □

Notice that, unlike other uses of potential functions in mistake-bound proofs, we do not make any claims about the relation between the value of the potential function used here and the distance between the actual weight vector $\mathbf{w}_r$ and a weight vector for the target. Indeed, we do not see any obvious relation between the value of this potential function and any measure of distance between $\mathbf{w}_r$ and a weight vector for the target.

6.2.2 Revising k-PDNF Fromulas

In this chapter we discuss Algorithm Rev-k-PDNF (see Figure 9), the revision algorithm for k-PDNFs. It has the same two-level structure that was also used by Valiant for learning PDNFs [128], but it uses different initial weights in the individual copies of Winnow (as it was discussed in the previous subsection). It also requires some variant of RevWinn applicable for conjunctions (which can be obtained by an easy transformation from RevWinn as explained at the beginning of the previous subsection, retaining the mistake bound described in Theorem 6.2); denote it RevWinnC and denote by InitC, hC, and UpdateC its main functions.

To fill up the details: Rev-k-PDNF consists of a top-level RevWinn algorithm that handles the selection of the appropriate projections. On the lower level, instances of RevWinnC are run, one for each of the $2^k\binom{n}{k}$ projections, to find the appropriate term for that particular projection. We call this the ρ **instance of** RevWinnC, and denote its weight vector by $\mathbf{w}^\rho$. The input resp. the label for each of these RevWinnC instances are $\mathbf{x}_r$ and y_r. An update is applied to the ρ instance of RevWinnC only when $\rho(\mathbf{x}_r) = 1$ (and additionally the top-level algorithm's prediction of the label was wrong and agreed with the prediction of the ρ-instance of RevWinnC), because in this case, by Equation (6.2) if ρ appears in the target formula with t-part t, then the output of the target formula agrees with t—and this is the key to the whole algorithm. Intuitively, we hope that for each term of the form $(\rho \cdot t)$ in the target formula, where ρ is a k-projection, the hypothesis of the ρ instance of RevWinnC will converge to t. The prediction of the ρ instance of RevWinnC is denoted $\hat{y}^\rho$ and $\hat{y}^\rho_r = \mathtt{hC}(\mathbf{x}_r, \mathbf{w}^\rho_{r-1})$.

For the top level, introduce a new Boolean variable v_ρ for each k-projection, and consider an instance of RevWinn run over these variables. In the rest of this section, $\mathbf{w}$ is used to denote the weight vector of this top level RevWinn instance (and, if we want to emphasize the round, $\mathbf{w}_r$ denotes its value *after* round r). The input for the top level is denoted $\mathbf{u}$; its value in round r, denoted $\mathbf{u}_r$, is defined by

$$\mathbf{u}_r(v_\rho) = \rho(\mathbf{x}_r) \wedge \mathtt{hC}(\mathbf{x}_r, \mathbf{w}^\rho_{r-1}).$$

The output of the top level in round r is

$$\hat{y}_r = \mathtt{h}(\mathbf{u}_r, \mathbf{w}_{r-1}).$$

The top-level `RevWinn` algorithm learns a disjunction over variables $newvar_\rho$, which would ideally consist of exactly those variables that are indexed by projections appearing in the target formula.

Algorithm 9 The procedure `Rev-`k`-PDNF`(φ, V).

```
1: {$\varphi = \rho_1 t_1 \vee \cdots \vee \rho_\ell t_\ell$ is the $k$-PDNF to be revised.}
2: $\mathbf{w} := \mathtt{Init}\ (\{v_\rho : \rho$ is a $k$-projection over V$\}, v_{\rho_1} \vee \cdots \vee v_{\rho_\ell})$
3: for each $k$-projection $\rho$ over V do
4:    if $\rho = \rho_i$ for some $i \in \{1, \ldots, s\}$ then
5:       $\mathbf{w}^\rho := \mathtt{InitC}(V, t_i)$
6:    else
7:       $\mathbf{w}^\rho := \mathtt{InitC}(V, \top)$
8:    end if
9: end for
10: for round $r = 0, 1, 2, \ldots$ with input $\mathbf{x}_r$ do
11:    Let $\mathbf{u}(v_\rho) := \rho(\mathbf{x}_r) \wedge \mathtt{hC}(\mathbf{x}_r, \mathbf{w}^\rho)$ for each $k$-projection $\rho$
12:    Output prediction $\hat{y}_r := \mathtt{h}(\mathbf{u}, \mathbf{w})$
13:    if receiving label $y_r$ for $\mathbf{x}_r$ it holds that $\hat{y}_r \neq y_r$ then
14:       {The top level algorithm made a mistake}
15:       $\mathbf{w} := \mathtt{Update}(y_r, \mathbf{u}, \mathbf{w})$
16:       for each $k$-projection $\rho$ with $\rho(\mathbf{x}_r) == 1$ and $\mathbf{u}_r(v_\rho) \neq y_r$ do
17:          $\mathbf{w}^\rho := \mathtt{UpdateC}(y_r, \mathbf{x}_r, \mathbf{w}^\rho)$
18:       end for
19:    end if
20: end for
```

Theorem 6.3 *Suppose that the initial and target formulas are, respectively, the k-PDNF$_n$ formulas*

$$\begin{aligned}\varphi &= \rho_1 t_1 \vee \cdots \vee \rho_\ell t_\ell \vee \rho_{\ell+1} t_{\ell+1} \vee \cdots \vee \rho_{\ell+s} t_{\ell+s},\\ \psi &= \rho_1 t_1^* \vee \cdots \vee \rho_\ell t_\ell^* \vee \rho_1' t_1' \vee \cdots \vee \rho_a' t_a',\end{aligned}$$

and $\hat{e} = \mathrm{dist}(\varphi, \psi)$. Then algorithm `Rev-`$k$`-PDNF` *makes $O(\hat{e} k \log n)$ mistakes.*

Proof

The top-level `RevWinn` revises a disjunction over the v_ρ's. There will be two sources of mistakes. First, the initial disjunction is not correct; it needs revising. Second, the values assigned to the v_ρ variables will sometimes be erroneous, because the low-level `RevWinnC`'s are imperfect—that is, $\mathbf{u}_r(v_\rho) \neq \rho(\mathbf{x}_r) \wedge t(\mathbf{x}_r)$ might occur in some round r for some term $(\rho \cdot t)$ of ψ. (The actual input $\mathbf{x}_r$ and classification y_r are assumed to be noiseless—that is, $y_r = \psi(\mathbf{x}_r)$ is assumed.)

Theorem 6.2 tells us how to calculate the overall number of mistakes of the top-level RevWinn as a function of three quantities: the revision distance, which is $s+a$, the total number of variables, both relevant and irrelevant for the disjunction, which is $2^k\binom{n}{k}$, and the total number of attribute errors, which we will now calculate.

In fact, we will *not* count all the attribute errors. We will count (actually provide an upper bound on) only those attribute errors that occur when RevWinn is charged with a mistake.

For $i = 1, \ldots, \ell$, the RevWinnC instance corresponding to projection ρ_i predicts $\hat{y}_r^{\rho_i} = \mathtt{hC}(\mathbf{x}_r, \mathbf{w}_{r-1}^{\rho_i})$ in round r. That RevWinnC instance updates for a mistake only when the overall algorithm makes a mistake (i.e., $\hat{y_r} \neq y_r$), its prediction was different from y_r (i.e., $\hat{y_r} \neq \hat{y}_r^{\rho_i}$), and $\rho_i(\mathbf{x}_r) = 1$. Now $y_r = \psi(\mathbf{x}_r) = t_i^*(\mathbf{x}_r)$ (the last equation holds because of projectivity and because $\rho_i(\mathbf{x}_r) = 1$). This means that the mistake bound for this RevWinnC tells us how many times this RevWinnC can make errors on rounds when the overall algorithm makes an error; after that number of mistakes, this RevWinnC will then always predict correctly. According to the discussion at the beginning of this subsection the mistake bound on this RevWinnC is $O(|t_i \triangle t_i^*| \ln n)$.

For $j = 1, \ldots, a$ a similar argument shows that there are at most $O(|t'_j| \ln n)$ rounds r where $\mathbf{u}_r(v_{\rho'_j}) \neq \rho'_j(\mathbf{x}_r) \wedge t'_j(\mathbf{x}_r)$ and the top-level RevWinn makes a mistake. Put $F(\varphi, \psi) = \left(\sum_{i=1}^{\ell} |t_i \triangle t_i^*| + \sum_{j=1}^{a} |t'_j|\right) \ln n$.

How many times can Rev-k-PDNF err when predicting? We just argued that the total number of attribute errors that occur when the top-level RevWinn makes a mistake is $O(F(\varphi, \psi))$. The total number of variables that the top-level RevWinn is working with is $2^k\binom{n}{k}$. Thus, the overall mistake bound is, by Theorem 6.2, $O\left(F(\varphi, \psi) + (s+a)\log\left(2^k\binom{n}{k}\right)\right) = O(\hat{e}k\log n)$, since $F = O(\hat{e}\log n)$.

Remark 6.1
For learning from scratch a k-PDNF$_n$ consisting of m terms, that is, for revising the empty k-PDNF$_n$ to a target k-PDNF$_n$, this algorithm has the same asymptotic mistake bound as Valiant's learning algorithm [128]: $O(kms\log n)$, where s is the maximum number of variables in any term in the target.

6.3 Exclusion Dimension

The combinatorial parameter, exclusion dimension of formula classes (for the definition see below) is in close connection with the query complexity of the given formula class (see, e.g. [11]). As the revision algorithm for projective DNFs works in the mistake bounded model, it seems interesting to discuss this parameter for this class. In this section we follow the terminology of Angluin [11]. (With minor variations, exclusion dimension is called unique specification dimension by Hegedűs [65] and certificate size by Hellerstein et al. [67].)

Let f be an n-variable Boolean function. A set $A \subseteq \{0,1\}^n$ is a **specifying set** of f with respect to a class $\mathcal{C}$ of Boolean functions if there is at most one function in $\mathcal{C}$ that agrees with f on A. (So clearly $\{0,1\}^n$ is always a specifying set.) The **specifying set size** of f with respect to $\mathcal{C}$ is

$$\mathrm{spec}_{\mathcal{C}}(f) = \min\{|A| : A \text{ is a specifying set for } f \text{ with respect to } \mathcal{C}\},$$

and the **exclusion dimension** of the class $\mathcal{C}$ is

$$\mathrm{XD}(\mathcal{C}) = \max\{\mathrm{spec}_{\mathcal{C}}(f) : f \notin \mathcal{C}\}.$$

A specifying set A for $f \notin \mathcal{C}$ such that *no* function in $\mathcal{C}$ agrees with f on A is also called a **certificate of exclusion** (or simply **certificate**) for f with respect to $\mathcal{C}$. In our constructions below, we will usually give certificates of exclusion, which clearly give upper bound for the specifying set size.

For the rest of this chapter specifying sets are always with respect to k-PDNF, so we write $\mathrm{spec}(f)$, omitting the subscript $\mathcal{C}$.

A function f is **minimally non-k-projective** if it is not k-projective, but any f' with $\mathcal{T}(f') \subset \mathcal{T}(f)$ is k-projective.

Proposition 6.4 *If f is minimally non-k-projective, then* $\mathrm{spec}(f) \geq |\mathcal{T}(f)| - 1$.

Proof

Suppose $|A| \leq |\mathcal{T}(f)| - 2$ for some $A \subseteq \{0,1\}^n$. Let $\mathbf{x}, \mathbf{y} \in \mathcal{T}(f) \setminus A$ be two different assignments. As f is minimally non-k-projective, there is $g_{\mathbf{x}} \in k$-PDNF$_n$ (resp. $g_{\mathbf{y}} \in k$-PDNF$_n$) such that $\mathcal{T}(g_{\mathbf{x}}) = (A \cap \mathcal{T}(f)) \cup \{\mathbf{x}\}$ (resp. $\mathcal{T}(g_{\mathbf{y}}) = (A \cap \mathcal{T}(f)) \cup \{\mathbf{y}\}$). Now $g_{\mathbf{x}}$ and $g_{\mathbf{y}}$ are different elements of k-PDNF$_n$ that agree with f on A, thus A is not a specifying set for f. □

We now present a lower and an upper bound for the exclusion dimension of k-PDNF$_n$, which show that for fixed k the exclusion dimension is $\Theta(n^k)$. We begin with a lemma that characterizes k-PDNF, give some examples, and then continue to the main theorem of this section that gives the bound.

Lemma 6.5 (a) *A function f is k-projective if and only if for every $\mathbf{x} \in \mathcal{T}(f)$ there is a k-conjunction ρ such that $\mathbf{x} \in \mathcal{T}(\rho)$ and $\mathcal{T}(f) \cap \mathcal{T}(\rho)$ is a cube.*

(b) *If for every $\mathbf{x} \in \mathcal{T}(f)$ there is a k-conjunction ρ such that $\mathcal{T}(f) \cap \mathcal{T}(\rho) = \{\mathbf{x}\}$, then f is k-projective.*

Proof

We show only (a), as (b) follows directly from (a). If f is k-projective then it can be written as $\varphi = \rho_1 t_1 \vee \cdots \vee \rho_\ell t_\ell$. Consider an $\mathbf{x} \in \mathcal{T}(f)$. Then $\rho_i t_i(\mathbf{x}) = 1$ for some i, thus $\mathbf{x} \in \mathcal{T}(\rho_i)$. The definition of PDNF implies that $\mathcal{T}(f) \cap \mathcal{T}(\rho_i) = \mathcal{T}(\rho_i t_i)$, which is a cube.

For the other direction, let us assume that for every $\mathbf{x} \in \mathcal{T}(f)$ there is a k-projection $\rho_{\mathbf{x}}$ such that $\mathbf{x} \in \mathcal{T}(\rho_{\mathbf{x}})$ and $\mathcal{T}(f) \cap \mathcal{T}(\rho_{\mathbf{x}}) = Q_{\mathbf{x}}$ is a cube. Then $Q_{\mathbf{x}}$ can be written as $\mathcal{T}(\rho_{\mathbf{x}} t_{\mathbf{x}})$ for some conjunction $t_{\mathbf{x}}$, and f can be written as the k-PDNF expression $\bigvee_{\mathbf{x} \in \mathcal{T}(f)} \rho_{\mathbf{x}} t_{\mathbf{x}}$. □

We illustrate Lemma 6.5 with the following example. We claim that the function $f(v_1, v_2, v_3, v_4) = v_1v_2 \vee v_3v_4$ is not 1-projective. Call an assignment that violates condition (a) in the lemma k-**deviant**, or simply **deviant**. It suffices to show that $\mathbf{1}$ is deviant. For symmetry reasons, we only need to show that $\mathcal{T}(f) \cap \mathcal{T}(v_1)$ is not a cube. Indeed, it contains $\mathbf{x}_1 = (v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 0, v_4 \mapsto 1)$ and $\mathbf{x}_2 = (v_1 \mapsto 1, v_2 \mapsto 0, v_3 \mapsto 1, v_4 \mapsto 1)$, but it does not contain their meet, $\mathbf{x}_1 \wedge \mathbf{x}_2 = (v_1 \mapsto 1, v_2 \mapsto 0, v_3 \mapsto 0, v_4 \mapsto 1)$.

Proposition 6.6 *For every k and $n \geq k + 2$ there is a non-k-projective function with $|\mathcal{T}(f)| = k + 3$.*

Proof
Let $\mathcal{T}(f) = \{\mathbf{1}_{\{i\}} : 1 \leq i \leq k+2\} \cup \{\mathbf{0}\}$. Then $\mathbf{0}$ is k-deviant, as every k-conjunction ρ satisfied by $\mathbf{0}$ contains at least two $\mathbf{1}_{\{i\}}$'s, but $\mathcal{T}(f) \cap \mathcal{T}(\rho)$ does not contain the join of these two assignments, and thus it cannot be a cube according to Proposition 2.1. □

The proposition gives a $(k+3)$-term-DNF function which is not k-projective.

Theorem 6.7 *1. For all n and k,*

$$\mathrm{XD}(k\text{-PDNF}_n) \leq 3\binom{n}{k} + 1,$$

and

2. if $n \geq 4k(k+1)$, then

$$\mathrm{XD}(k\text{-PDNF}_n) \geq \binom{\lfloor n/4 \rfloor}{k} - 1.$$

Proof
For the upper bound, we will calculate an upper bound on the size of a certificate of exclusion for any $f \notin k$-PDNF$_n$ with respect to k-PDNF$_n$.

To show that a a function f is not k-projective, it suffices to present a deviant assignment $\mathbf{x}$ (i.e., $\mathbf{x}$ violates Condition (a) of Lemma 6.5) together with a certificate of $\mathbf{x}$'s deviance. For the certificate of $\mathbf{x}$'s deviance it suffices to specify, according to Proposition 2.1, for every k-conjunction ρ with $\rho(\mathbf{x}) = 1$, three assignments $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ such that $\rho(\mathbf{x}_1) = \rho(\mathbf{x}_2) = \rho(\mathbf{x}_3) = 1$, $\mathbf{x}_1 \wedge \mathbf{x}_2 \leq \mathbf{x}_3 \leq \mathbf{x}_1 \vee \mathbf{x}_2$ and $f(\mathbf{x}_1) = f(\mathbf{x}_2) = 1$, $f(\mathbf{x}_3) = 0$. The number of k-conjunctions with $\rho(\mathbf{x}) = 1$ is $\binom{n}{k}$. Thus the upper bound follows: 1 for $\mathbf{x}$ itself, and then 3 assignments each for at worst all of the k-conjunctions.

For the lower bound, in view of Proposition 6.4, it is sufficient to construct a minimally non-k-projective n-variable function $f_{n,k}$ that takes the value 1 at many points. First we describe the construction in the case when n is even and $k = 1$. Let $n = 2s$, let $\hat{\mathbf{a}} = \mathbf{1}^{(v_1 \mapsto 0, \ldots, v_s \mapsto 0)}$ for $i = 1, \ldots, s$, and define $f_{n,k}$ by $\mathcal{T}(f_{n,k}) = \{\mathbf{a}_i := \hat{\mathbf{a}}^{(v_i \mapsto 1, v_{s+i} \mapsto 0)} : i = 1, \ldots, s\} \cup \{\mathbf{0}\}$. We claim that $f_{n,k}$ is minimally non-1-projective.

The non-1-projectivity of $f_{n,k}$ follows from the fact that $\mathbf{0}$ is deviant: any 1-projection ρ containing $\mathbf{0}$ must be a negative literal, and thus it contains some assignment(s) $\mathbf{a}_i$, but it does not contain any assignment of positive weight less than s. Thus, by the remark following Proposition 2.1, $\mathcal{T}(f_{n,k}) \cap \mathcal{T}(\rho)$ is not a cube. On the other hand, the $\mathbf{a}_i$'s are *not* deviant for $f_{n,k}$. This holds as they satisfy the condition of part (b) of Lemma 6.5: the 1-conjunction v_{s+i} contains only $\mathbf{a}_i$ from $\mathcal{T}(f_{n,k})$. Now we show that every f' with $\mathcal{T}(f') \subset \mathcal{T}(f_{n,k})$ is 1-projective. Indeed, if $f'(\mathbf{0}) = 0$ then this follows from part (b) of Lemma 6.5 directly. Otherwise the only thing to note is that if $f'(\mathbf{a}_i) = 0$, then the 1-conjunction $\overline{v_i}$ contains only $\mathbf{0}$ from $\mathcal{T}(f')$.

For the construction in the general case we use the following lemma. In the lemma we consider $\{0,1\}^p$ to be the p-dimensional vector space over $GF(2)$ and I to be the $p \times p$ identity matrix.

Lemma 6.8 *Let A be a $p \times p$ 0–1 matrix such that both A and $A \otimes I$ are nonsingular. Assume that $k(k+1) < 2^p$ and define the mapping*

$$h(\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}) = \{\mathbf{b}_1 \otimes A\mathbf{b}, \ldots, \mathbf{b}_k \otimes A\mathbf{b}\},$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_k$ are different elements of $\{0,1\}^p$, and $\mathbf{b} = \mathbf{b}_1 \otimes \cdots \otimes \mathbf{b}_k$. Then it holds that

(a) *h is a bijection, and*

(b) *for every $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ and $\mathbf{d}_1, \ldots, \mathbf{d}_k$ there is a $\mathbf{b}_k$ different from $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$, such that the elements of $h(\{\mathbf{b}_1, \ldots, \mathbf{b}_k\})$ are all different from the $\mathbf{d}_i$'s.*

Proof
If $h(\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}) = \{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$, then $\mathbf{d}_1 \otimes \cdots \otimes \mathbf{d}_k = \mathbf{b} \otimes (k \bmod 2)A\mathbf{b}$, which is equal to $\mathbf{b}$ (resp., $(A \otimes I)\mathbf{b}$), if k is even (resp., odd). Thus, knowing $\mathbf{d}_1, \ldots, \mathbf{d}_k$ we can first determine $\mathbf{b}$, and then we can determine every $\mathbf{b}_i$ by $\mathbf{b}_i = \mathbf{d}_i \otimes A\mathbf{b}$. Hence h is injective, and thus it is also bijective.

For (b), note that a value for $\mathbf{b}_k$ can fail to satisfy the requirement only if it is either equal to one of the $\mathbf{b}_i$'s, or if $\mathbf{b}_i \otimes A\mathbf{b} = \mathbf{d}_j$ for some $1 \le i, j \le k$. In each case we can solve for $\mathbf{b}_k$, thus there are altogether at most $k + k^2$ bad choices. Choosing any of the other $2^p - (k + k^2)$ vectors meets our requirements for $\mathbf{b}_k$. □

Now we continue the proof of Theorem 6.7 with the general case $k > 1$. First, we need a matrix that fulfills the conditions of Lemma 6.8. It is easily verified that, for example, the matrix A with all 0's except $a_{1,1} = a_{p,1} = a_{i,i+1} = 1$ (where $i = 1, \ldots, p-1$) is such a matrix. It is clear from the definition of h that if the $\mathbf{b}_i$'s are all different, then $h(\{\mathbf{b}_1, \ldots, \mathbf{b}_s\})$ also consists of s different elements.

Now let $p = \lfloor \log \frac{n}{2} \rfloor$, and put $s = 2^p$. If I is a k-element subset of $\{1, 2, \ldots, s\}$, put $\hat{\mathbf{a}} := \mathbf{0}^{(v_{s+1} \mapsto 1, \ldots, v_{2s} \mapsto 1)}$, define $\alpha_I := \mathbf{0}_{\{v_i : i \in I\}}$ and $\beta_I := \mathbf{1}_{\{v_{s+i} : i \in I\}}$ [3], and put $\mathbf{a}_I = \hat{\mathbf{a}}^{(\alpha_I, \beta_I)}$ and define $f_{n,k}$ by $\mathcal{T}(f_{n,k}) = \{\mathbf{a}_I : I \subseteq \{1, 2, \ldots, s\}, |I| = k\} \cup \{\mathbf{0}\}$.

[3]With a slight abuse of notation the $\mathbf{b}_i$ vectors are used both to denote elements of $\{1, 2, \ldots, s\}$ and their binary representations.

We claim that $f_{n,k}$ is minimally non-k-projective. The argument for this is very similar to the argument in the special case above. The projection $\rho_I = \bigwedge_{i \in h(I)} v_{s+i}$ contains only $\mathbf{a}_I$ from $\mathcal{T}(f_{n,k})$ by part (a) of Lemma 6.8, and if $\mathbf{a}_I$ is not contained in $\mathcal{T}(f')$ for some f' with $\mathcal{T}(f') \subseteq \mathcal{T}(f_{n,k})$, then the projection $\rho_0 = \bigwedge_{i \in I} \overline{v_i}$ contains only $\mathbf{0}$ from $\mathcal{T}(f')$. It only needs to be shown that $\mathbf{0}$ is deviant for $f_{n,k}$. Let ρ be any k-conjunction containing $\mathbf{0}$. We can assume that every literal $\overline{v_i}$ in ρ has $i \leq 2s$, as the other literals do not exclude any $\mathbf{a}_I$. We show that besides $\mathbf{0}$ there is an $\mathbf{a}_I$ in $\mathcal{T}(\rho)$, which implies the claim by the remark following Proposition 2.1. If all the literals come from the first s variables then $\mathbf{a}_I$ corresponding to these literals clearly satisfies the requirements. Otherwise, let us assume that the literals in ρ are of the form $\overline{v_i}$, for $i \in I_1 \cup I_2$, $I_1 \subseteq \{1, 2, \ldots, s\}, I_2 \subseteq \{s+1, s+2, \ldots, 2s\}$, $|I_2| > 0$ and $|I_1| + |I_2| = k$. By part (b) of Lemma 6.8 there is an $I \subseteq \{1, 2, \ldots, s\}, |I| = k, I_1 \subset I$ such that $h(I) \cap I_2 = \emptyset$, and by definition, $\mathbf{a}_I \in \mathcal{T}(\rho)$. □

Using the results on the relation between the exclusion dimension and the complexity of learning with membership and proper equivalence queries [11; 65; 67] we get the following.

Proposition 6.9 *The class k-PDNF$_n$ can be learned with $O\left(n\, 2^k \binom{n}{k}^2\right)$ membership and proper equivalence queries. On the other hand the query complexity of this class is at least $\binom{\lfloor n/4 \rfloor}{k} - 1$.*

Proof
The query complexity of a formula class $\mathcal{R}$ is at most $\mathrm{XD}(\mathcal{R}) \cdot \log |\mathcal{R}|$ and at least $\mathrm{XD}(\mathcal{R})$ (see, e.g., [11]). We are interested in the case when $\mathcal{R}$ is the set of k-PDNFs. Since the number of k-conjunctions over n variables is $\binom{n}{k} 2^k$ (choose k variable from the n and then choose an orientation for each), a k-PDNF consists of at most $2^k \binom{n}{k}$ terms. Noting that the number of K-term-DNFs is at most $(3^n)^K$, one derives the upper bound $3^{n 2^k \binom{n}{k}}$ for the number of k-PDNFs which, combined with Theorem 6.7, completes the proof. □

The number of queries used by the learning algorithm that the above proposition referres to, is polynomial in n for every fixed k. On the other hand, the *running time is not necessarily polynomial.*

Blum [22], using ideas from Littlestone and Helmbold et al. [69; 91], shows that a simple subclass of decision lists (called 1-decision lists) is efficiently learnable in the mistake-bounded model. It follows from a straightforward generalization of this result and Proposition 4 in [115] (discussing the relation of projective DNFs and decision lists) that for every fixed k, the class k-PDNF is learnable with polynomially many *improper* equivalence queries and with polynomial running time. (Yet another proof for this is Theorem 6.2: evidently, efficient learnability follows from efficient revision.)

Thus the question wether the class can be learned with *proper* equivalence queries in *polynomial running time* is still open.

6.4 Concluding Remarks

As mentioned, an interesting direction would be to study the computational complexity of algorithmic questions related to PDNF. Recall that the discussed results leave open the question whether there is a computationally efficient equivalence and membership query learning algorithm for k-PDNF.

Another direction could be to consider noisy model, that is, when in some round r the label y_r is not the correct classifiaction of instance $\mathbf{x}_r$, that is, $y_r \neq f_{\mathrm{trg}}(\mathbf{x}_r)$ (as in [113]). A special motivation for this is that, for technical reasons, we had already considered noise in the intermediate steps in the analysis of algorithm `Rev-`k`-PDNF`. However this model does not seem to be too interesting. Assume that some algorithm `Algo` is an efficient learning algorithm for some formula (or concept) class with mistake bound MB when noise is not allowed. Then this algorithm can be used to learn the same class in noisy environment making at most $\mathrm{MB} \cdot \mathrm{FL}$ mistakes, where FL denotes the number of false labels (i.e., the number of rounds when $y_r \neq f_{\mathrm{trg}}(\mathbf{x}_r)$) in a given run, iterating the following: initialize algorithm `Algo`, run it as long as its mistake bound is below MB, then reset. (Note also that if FL and/or MB is not known in advance, one can use the usual doubling technique—but this adds an extra logarithmic factor.) For more on this issue and some other related topics see for example [20; 21].

Finally note that the results presented in this chapter—unless noted otherwise—appeared in the paper [115], co-authored by the author of the present dissertation.

Part II

Characterization Results

Chapter 7

1-PDNF Formulas

Chapter 6 discussed the revision of the k-PDNF formulas, the class introduced by Valiant [128] motivated by certain biological considerations. During the research aimed to analyze this apparently new class, a special subclass, the 1-PDNF formulas have shown some interesting regularities in their form. Further examination of this phenomenon has confirmed that this was not just a mere coincidence, and indeed there is some nice characterization for the class of 1-PDNFs. In this chapter this result is presented. Throughout the chapter the notations and terminology introduced in Chapter 6 are used.

7.1 p-irredundancy and a Characterization of 1-PDNF Formulas

First let us note that if φ is a 1-PDNF that includes two complementary projections, that is, it is of the form $\varphi = vt_1 \vee \overline{v}t_2 \vee \cdots$ for some variable v, then by deleting everything else besides these two terms, we get an equivalent formula. Indeed, by Equation (6.2) $vt_1 \vee \overline{v}t_2 \equiv v\varphi \vee \overline{v}\varphi$, which is obviously equivalent to φ.

We formulate a notion of irredundancy for 1-PDNF, which we call p-irredundancy to distinguish it from the usual notion of irredundancy for DNF. Unlike the standard notion, p-irredundancy of a 1-PDNF is easy to decide.

Definition 7.1 *A 1-PDNF formula* $\varphi = \rho_1 t_1 \vee \cdots \vee \rho_\ell t_\ell$ *is* **p-irredundant** *if the following conditions all hold:*

(a) $\mathrm{Lit}(\rho_i t_i) \not\subseteq \mathrm{Lit}(\rho_j t_j)$ *for each distinct* $i, j \in \{1, \ldots, \ell\}$,

(b) $\rho_i, \overline{\rho_i} \notin \mathrm{Var}(t_i)$ *for every* $1 \leq i \leq \ell$,

(c) *if* $\ell \geq 3$ *then* $\rho_i \neq \overline{\rho_j}$ *for each distinct* $i, j \in \{1, \ldots, \ell\}$.

Otherwise, φ *is called* **p-redundant**.

The first condition says that no term implies another, the second that in each term the projection and conjunction parts are disjoint (a formula violating any of these two conditions has a trivial simplification), and the third that if there are at least three terms, then no two projections are complementary (recall the argument above).

Given a 1-PDNF expression, one can easily transform it into a p-irredundant form as follows. First delete any term that has the negation of its ρ-part contained in its t-part (violating (b)). Next check if there are two complementary projections, and if there are, then delete all the other terms, thereby guaranteeing (c) (again, recall the argument from the beginning of the section). Otherwise, delete every term subsumed by another term, ensuring (a). Finally, if in a remaining term the t-part contains the projection literal, then delete the projection literal from that term. The final expression is a p-irredundant 1-PDNF, which is equivalent to the original one.

The above algorithm runs in polynomial time, thus we have:

Proposition 7.2 *There is a polynomial algorithm which, given a 1-PDNF expression, transforms it into an equivalent p-irredundant 1-PDNF expression.*

In view of this it thus suffices to consider only 1-PDNF expressions in p-irredundant form for the characterization of 1-PDNF formulas:

Theorem 7.3 *A formula φ is a p-irredundant 1-PDNF formula if and only if it is either of the form*

$$\varphi = \bigvee_{i=1}^{s} (\rho_{i,1} t_i \vee \cdots \vee \rho_{i,\ell_i} t_i),$$

where $\rho_{i,r} \notin \mathrm{Var}(t_i)$ and $\overline{\rho_{i,r}} \in \mathrm{Lit}(t_j)$ for every distinct $i, j \in \{1, \ldots, s\}$ and $1 \leq r \leq \ell_i$, and furthermore the projections are all based on different variables, or it is of the form

$$\varphi = vt \vee \overline{v}t' \ ,$$

where $v \notin \mathrm{Var}(t)$ and $\overline{v} \notin \mathrm{Var}(t')$.

Informally, the first case of the theorem says the following. Let us consider a term in a p-irredundant 1-PDNF to consist of a "stem" t and a "petal" ρ. Then the petal of each term is not included in its stem (that much is clear from the definition of p-irredundancy) and if two terms have different stems then each stem contains the negation of the other one's petal. In other words, each stem consists of the negations of all the petals corresponding to terms with different stems, plus, possibly, some other literals.

7.2 Proof of Theorem 7.3

First we give a description of those p-irredundant 1-projective DNF that represent either a monotone or an $\mathbf{a}$-unate function, and then we give the general description. We assume *w.l.o.g.* throughout this section that each 1-PDNF in question determines a non-constant function and has terms that do not contain any complementary literals. Throughout the proof we also frequently use the fact that for arbitrary terms t and t' it holds that $\mathcal{T}(t) \subseteq \mathcal{T}(t')$ if and only if $\mathrm{Lit}(t') \subseteq \mathrm{Lit}(t)$ (see Section 2.3).

Lemma 7.4 *A formula φ is a p-irredundant 1-PDNF formula representing a monotone (resp. $\mathbf{a}$-unate) function if and only if it is either of the form*

$$\varphi = \rho_1 t \vee \cdots \vee \rho_\ell t, \tag{7.1}$$

where $\rho_1, \ldots, \rho_\ell$ are different unnegated variables (resp. literals whose signs agree with $\mathbf{a}$) not contained in $\mathrm{Var}(t)$, and t is a monotone (resp. $\mathbf{a}$-unate) term, or it is of the form

$$\varphi = \rho t \vee \overline{\rho} t t', \tag{7.2}$$

where ρ is an unnegated variable (resp. its sign agrees with $\mathbf{a}$) and t, t' are monotone (resp. $\mathbf{a}$-unate) terms not containing ρ or $\overline{\rho}$.

Proof

We prove only the monotone case, as the $\mathbf{a}$-unate case follows by considering the monotone function obtained by replacing assignment $\mathbf{x}$ with $\mathbf{x} \otimes \mathbf{a}$. (Note that a function f is k-PDNF if and only if $f_\mathbf{a}$ is, where $f_\mathbf{a}(\mathbf{x}) = f(\mathbf{x} \otimes \mathbf{a})$.) It follows directly from the definitions that every expression of the form of Equation (7.1) or (7.2) is indeed a p-irredundant 1-PDNF expression.

Let φ be an arbitrary monotone p-irredundant 1-PDNF formula. Separating the negated and unnegated projections, *w.l.o.g.* let us write φ as

$$\varphi = \bigvee_{i \in I} (v_i \cdot t_i) \vee \bigvee_{j \in J} (\overline{v_j} \cdot t_j). \tag{7.3}$$

(This representation of φ is convenient for the following series of claims.)

Claim 7.5 *For any monotone formula φ of the form as in Equation (7.3) it holds that the index set I is nonempty, and that t_r is monotone for all $r \in I \cup J$.*

Proof

The first part of the Claim holds because φ determines a non-constant monotone function, thus $\varphi(\mathbf{1}) = 1$.

To prove monotonicity for t_i, $i \in I$, note that $\mathbf{1}$ satisfies every monotone projection, thus by projectivity $(v_i \cdot t_i)(\mathbf{1}) = \varphi(\mathbf{1})$, which equals 1 (as argued above), thus t_i must be monotone.

Finally, let us consider a term $\overline{v_j} t_j$ with $j \in J$. Asssume for the contradiction that term t_j contains negative literal $\overline{v_r}$. Let $\mathbf{x}$ be any assignment satisfying the term $\overline{v_j} \cdot t_j$ and thus φ. By monotonicity $\mathbf{x}^{(v_r \mapsto 1)}$ must satisfy φ. However, then, by projectivity and because $r \neq j$ (by (b) of p-irredundancy), $\mathbf{x}^{(v_r \mapsto 1)}$ must satisfy t_j, a contradiction. $\square$

Claim 7.6 *For any monotone formula φ of the form as in Equation (7.3) it holds that $\mathcal{T}(\varphi) \subseteq \mathcal{T}(t_i)$ for all $i \in I$.*

Proof
Pick an arbitrary $i \in I$. Let $\mathbf{x} \in \mathcal{T}(\varphi)$, so $\varphi(\mathbf{x}) = 1$. By monotonicity $\varphi\left(\mathbf{x}^{(v_i \mapsto 1)}\right) = 1$, by projectivity $t_i\left(\mathbf{x}^{(v_i \mapsto 1)}\right) = 1$, and by (b) of p-irredundancy $t_i(\mathbf{x}) = 1$, which proves the claim. □

Claim 7.6 can be used to show that the t-parts of the terms with positive ρ-parts are all the same—that is, $t_i = t$ for $i \in I$ for some term t:

Claim 7.7 *For any monotone formula φ of the form as in Equation (7.3) it holds that there must be a single term t such that*

$$\varphi = \bigvee_{i \in I} (v_i \cdot t) \vee \bigvee_{j \in J} (\overline{v_j} \cdot t_j).$$

Proof
Consider any two distinct $i, j \in I$. From projectivity and from Claim 7.6 it follows that $\mathcal{T}(v_i t_i) \subseteq \mathcal{T}(\varphi) \subseteq \mathcal{T}(t_j)$ and, likewise, that $\mathcal{T}(v_j t_j) \subseteq \mathcal{T}(\varphi) \subseteq \mathcal{T}(t_i)$. Thus

$$\mathrm{Lit}(t_j) \subseteq \mathrm{Lit}(v_i t_i) \text{ and } \mathrm{Lit}(t_i) \subseteq \mathrm{Lit}(v_j t_j). \tag{7.4}$$

From this and from (a) of p-irredundancy it follows that $v_j \notin \mathrm{Lit}(v_i t_i)$ and $v_i \notin \mathrm{Lit}(v_j t_j)$. But then $\mathrm{Lit}(t_j) = \mathrm{Lit}(t_i)$. □

Putting together Claims 7.5 and 7.7, it follows that we are done if $J = \emptyset$. The remaining case (i.e., when $J \neq \emptyset$) is handled by the following Claim.

Claim 7.8 *Let π be a monotone p-irredundant 1-PDNF formula of the form*

$$\pi = \bigvee_{i \in I} (v_i \cdot t) \vee \bigvee_{j \in J} (\overline{v_j} \cdot t_j),$$

where I and J are nonempty sets, furthermore t_j, for $j \in J$, and t are monotone terms. Then $\pi = v_i t \vee \overline{v_i} t t'$ for some variable v_i and some monotone term t'.

Proof
It follows from projectivity and from Claim 7.6 that $\mathcal{T}(\overline{v_j} t_j) \subseteq \mathcal{T}(\pi) \subseteq \mathcal{T}(t)$, thus $\mathrm{Lit}(t) \subseteq \mathrm{Lit}(\overline{v_j} t_j)$, and so (as t is monotone) $\mathrm{Lit}(t) \subseteq \mathrm{Lit}(t_j)$. Thus π can be written as

$$\pi = \bigvee_{i \in I} (v_i \cdot t) \vee \bigvee_{j \in J} (\overline{v_j} \cdot t t'_j),$$

where now $I, J \neq \emptyset$ and t, t'_j are monotone terms. If $I = J = \{i\}$ for some i, then we are done. For the rest of the proof we assume that this is not the case, and show that this leads to contradiction.

Now it must be the case, that there are terms $(v_i \cdot t)$ and $(\overline{v_j} \cdot t t'_j)$ in π such that $i \neq j$. Thus $\mathcal{T}(v_i \overline{v_j} t t'_j) \neq \emptyset$ (by (a) of p-irredundancy), and it also holds (by Equation (6.4)) that $\mathcal{T}(\overline{v_j} v_i t) = \mathcal{T}(v_i \overline{v_j} t t'_j)$. Then either $t'_j = v_i$ or $t'_j = \top$. But $t'_j = v_i$ would violate (a) of p-irredundancy, thus it must be that $t'_j = \top$.

Let us consider first the case when π contains only two terms. Then it must be of the form $\pi = (v_i \cdot t) \vee (\overline{v_j} \cdot t)$. Then, on one hand, if $v_j \notin t$, then it contradicts the monotonicity of π (in variable v_j), on the other hand, if $v_j \in t$, then it contradicts (b) of p-irredundancy.

Let us consider now the case when π has at least three terms. Since $t'_j = \top$, by projectivity $\mathcal{T}(\overline{v_j}t) \subseteq \mathcal{T}(\pi)$, and thus by monotonicity $\mathcal{T}(t) \subseteq \mathcal{T}(\pi)$. With Claim 7.6. this implies $\mathcal{T}(t) = \mathcal{T}(\pi)$. But then for every other $k \in J$ it holds that $\mathcal{T}(\overline{v_k}\pi) = \mathcal{T}(\overline{v_k}t)$, meanwhile by projectivity $\mathcal{T}(\overline{v_k}tt'_k) = \mathcal{T}(\overline{v_k}\pi)$, so $t'_k = \top$. Therefore

$$t \equiv \pi = \bigvee_{i \in I}(v_i \cdot t) \vee \bigvee_{j \in J}(\overline{v_j} \cdot t) \equiv \left(\bigvee_{i \in I} v_i \vee \bigvee_{j \in J} \overline{v_j} \right) t.$$

This can only hold if some variable occurs both in I and J, contradicting condition (c) of the definition of p-irredundancy for π.

This completes the proof of the claim. □

Now the lemma, as mentioned, follows from Claims 7.5, 7.7 and 7.8. □

The example of (6.3) (i.e., that $(x \cdot y) \vee (z \cdot y) \equiv (x \cdot y) \vee (\overline{x} \cdot yz)$) shows that the representation as a p-irredundant 1-PDNF is not always unique. Also, it is an interesting consequence of the theorem that there are monotone 1-PDNF functions, which cannot be written as a monotone 1-PDNF. Consider, for example, the 1-PDNF

$$(x \cdot 1) \vee (\overline{x} \cdot yz),$$

representing the monotone function $x \vee yz$. If there were an equivalent monotone 1-PDNF, then it could be transformed into a monotone p-irredundant 1-PDNF, which must look like the first case in the theorem. But then the minimal elements of $\mathcal{T}(x \vee yz)$ (where minimality is understood in the partial order defined by "$\leq$") must have Hamming distance at most 2, which is not the case for this function:

$$\mathrm{distH}((x \mapsto 1, y \mapsto 0, z \mapsto 0), (x \mapsto 0, y \mapsto 1, z \mapsto 1)) = 3\,.$$

Now we are ready to prove Theorem 7.3

Proof (of Theorem 7.3)
Again, one direction of the theorem follows immediately from the definition of p-irredundancy. For the other direction, if there are two complementary projections in φ, then by condition (c) of p-irredundancy, φ must be of the form $vt \vee \overline{v}t'$. Otherwise, let us assume that φ is of the form $\varphi = \rho_1 t_1 \vee \cdots \vee \rho_\ell t_\ell$. Consider any two terms $\rho_i t_i$ and $\rho_j t_j$. If $\mathcal{T}(\rho_i t_i) \cap \mathcal{T}(\rho_j t_j) \neq \emptyset$, then $\rho_i t_i \vee \rho_j t_j$ is unate, and by Lemma 7.4 it must be the case that $t_i = t_j$. On the other hand, if $\mathcal{T}(\rho_i t_i) \cap \mathcal{T}(\rho_j t_j) = \emptyset$, then by projectivity, it holds that $\mathcal{T}(\rho_i \rho_j t_j) = \emptyset$, thus $\overline{\rho_i} \in \mathrm{Lit}(t_j)$. Thus for every term $\rho_i t_i$, those terms $\rho_j t_j$ for which $\mathcal{T}(\rho_i t_i) \cap \mathcal{T}(\rho_j t_j) \neq \emptyset$ have the same conjunction part, and all the other terms contain $\overline{\rho_i}$ in their conjunction part. □

7.3 Concluding Remarks

The main result of this chapter is the characterization of the subclass of 1-PDNF functions. It would be interesting to get a description of k-PDNF functions for larger k.

Finally note that the results presented in this chapter—unless noted otherwise—appeared in the paper [115], co-authored by the author of the present dissertation.

Chapter 8

k-term-DNF Formulas with Largest Number of Prime Implicants

Prime implicants of a Boolean function (or, in other words, maximal subcubes of a subset of the n-dimensional hypercube [1]) form a basic concept for the theory of Boolean functions and their applications. Concerning the maximal number of prime implicants, it is known that an n-variable Boolean function can have at most $O\left(\frac{3^n}{\sqrt{n}}\right)$ prime implicants, and there are n-variable Boolean functions with $\Omega\left(\frac{3^n}{n}\right)$ prime implicants (see, e.g., [31]).

Another case considered is the maximal number of prime implicants of Boolean functions represented by disjunctive normal forms (DNF) with a bounded number of terms. The result that a k-term-DNF can have at most $2^k - 1$ prime implicants was discovered independently by Chandra and Markowsky [31], Levin [90] and McMullen and Shearer [97]. (For a recent application in computational learning theory, see Hellerstein and Raghavan [68].) It was shown by Laborde [88], Levin [90] and McMullen and Shearer [97] that the bound is sharp, i.e., there are k-term-DNFs with $2^k - 1$ prime implicants (Chandra and Markowsky gave an example with more than $2^{k/2}$ prime implicants). In view of these results, we call a DNF **maximal** if it has k terms and $2^k - 1$ prime implicants for some k.

In this chapter, on one hand, the above results of [31; 88; 90; 97] (about maximal DNFs) are presented, and on the other hand, these results get completed by determining all the maximal disjunctive normal forms.

[1]This and the following chapter heavily relies on the view discussed in Subsection 2.3.1: to view $\mathcal{A}(\mathcal{V}')$ as the n-dimensional cube, and a term as a subcube of it, where $\mathcal{V}'$ is some finite subset of $\mathcal{V}$.

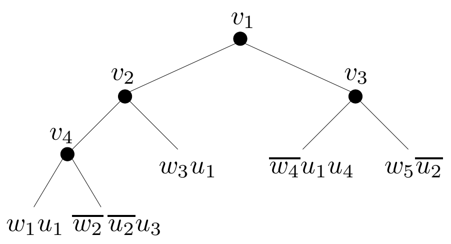


Figure 8.1: A non-repeating, unate-leaf decision tree (NUD). The labels of the edges are omitted for simplicity.

8.1 Nonrepeating Decision Trees and the Characterization of Maximal DNFs

In order to formulate the description, let us introduce the notion of non-repeating, unate-leaf decision tree.

For a given $k \geq 2$ and $r \geq 0$, let us consider the pairwise distinct variables $v_1, \ldots, v_{k-1}, w_1, \ldots, w_k$ and $u_1, \ldots, u_r$. For each of the w and u variables, pick an orientation, i.e., form the literals ε_i and δ_j, where ε_i is either w_j or $\overline{w_j}$ and δ_j is either u_j or $\overline{u_j}$, for $i = 1, \ldots, k$ and $j = 1, \ldots, r$. A **non-repeating unate-leaf decision tree** (NUD) T over these variables and literals is constructed by taking an LBT over variables $v_1, \ldots, v_{k-1}$ with $k-1$ inner nodes such that each inner node has different label, also assign to each leaf a distinct w literal from those formed above, and, in addition, assign to each leaf an arbitrary subset of the u literals formed above. The set of leaves of T is denoted by L. If we want to mention the number of v variables and w literals used in the construction, then we refer to T as a k-NUD (the value r is irrelevant). Figure 8.1 gives an example of a 5-NUD (the labeling of the edges is omitted for simplicity).

A k-NUD represents a k-term-DNF, determined as follows. For a leaf $\ell \in L$, let the term t_ℓ be the conjunction of

- the v literals along the path leading to ℓ, and of
- the w and u literals assigned to ℓ.

The k-term-DNF represented by the k-NUD T is

$$\varphi_T = \bigvee_{\ell \in L} t_\ell .$$

For example, the 5-term-DNF represented by the 5-NUD of Figure 8.1 is

$$\overline{v_1}\,\overline{v_2}\,\overline{v_4}\,w_1\,u_1 \;\vee\; \overline{v_1}\,\overline{v_2}\,v_4\,\overline{w_2}\,\overline{u_2}\,u_3 \;\vee\; \overline{v_1}\,v_2\,w_3\,u_1 \;\vee\; v_1\,\overline{v_3}\,\overline{w_4}\,u_1\,u_4 \;\vee v_1\,v_3\,w_5\,\overline{u_2}.$$

The Boolean function represented by φ_T can also be thought of in the following way: given a truth assignment $\mathbf{x}$ to all the variables, use the values of the v variables to determine a path from the root to a leaf. The function value is 1 if $\mathbf{x}$ makes all the

w and u literals assigned to this leaf true, and it is 0 otherwise. It is clear from the definition that the inputs accepted at a leaf ℓ are precisely those assignment which satisfy the term t_ℓ. The function φ_T is a generalized addressing function or multiplexer [109; 132]. If a DNF φ comes from a NUD T, then T can be reconstructed from φ. The w and u literals are those which are unate in φ, i.e., their negation does not occur in φ, while the v variables are those which occur both negated and unnegated. Among the v variables, the one labeling the root is the only one which occurs in every term (either unnegated or negated). The left child is the only v variable which occurs in every term containing the negation of the root variable, etc. In view of this correspondence, with some abuse of terminology, we can talk about a DNF being a NUD, rather than corresponding to a NUD. The maximal DNF of [88; 97] (resp., [90]) corresponds to a tree which is a single path (resp., a complete binary tree), without any u literals. A NUD generalizes these examples by allowing for an binary arbitrary tree and for the additional u literals. Now we can formulate the description of maximal DNF.

Theorem 8.1 *A DNF is maximal if and only if it corresponds to a NUD.*

A closely related class of DNF *tautologies* is obtained if we consider trees with the same kind of inner nodes, but without any literals assigned to the leaves. In the case of the example of Figure 8.1, the corresponding DNF tautology is

$$\overline{v_1}\,\overline{v_2}\,\overline{v_4} \lor \overline{v_1}\,\overline{v_2}\,v_4 \lor \overline{v_1}\,v_2 \lor v_1\,\overline{v_3} \lor v_1\,v_3\,.$$

Let us refer to this class of tautologies as **non-repeating decision tree** tautologies, or **ND**'s. The main step in the proof of Theorem 8.1, the ND Lemma (Lemma 8.11) is to show that for every DNF tautology the following two properties are equivalent: (a) any two of its terms have exactly one conflicting pair of literals (in other words, the terms are pairwise neighboring), (b) it is an ND. Lemma 8.11 was proven recently, independently from our work, by Kullmann [85; 86] [2]. Also note that Theorem 9.1 generalizes the result of the ND Lemma, thus the latter simple follows from the former; however the proof for the former case is more simple, and it seems to worth discussing it separately.

We note that ND's come up in other contexts as well, e.g., in connection with the complexity of analytic tableaux (Urquhart [125], referring to earlier unpublished work of Cook, and Arai *et al.* [15]).

The characterization of ND's as pairwise neighboring DNF tautologies is a direct consequence of the following *Splitting Lemma* (Lemma 8.10): if the n-dimensional

[2]Kullmann's proof uses the concept of Hermitian defect and other concepts from linear algebra. (The Hermitian rank of a symmetric matrix is the maximum of the number of positive and the number of negative eigenvalues of the matrix (Gregory, Watts and Shader [55]), and the Hermitian defect is the difference of the order of the matrix and its Hermitian rank [85; 86].) Kullmann also uses the characterization of ND's as strongly minimal tautologies with the additional property that the number of terms is one more than the number of variables (Aharoni and Linial [1], Davydov *et al.* [33], Kullmann [84]), proved using Hall's theorem or resolution techniques. (A tautology is strongly minimal if deleting any term, or adding any literal to a term results in a non-tautology.) Our proof is an elementary combinatorial argument.

hypercube is partitioned into subcubes of pairwise distance one, then there is a split of the whole cube into two half cubes such that every cube of the partition is contained in one of the two halves. Note that the result presented in the next chapter (Theorem 9.1) generalizes this result; however the proof for is much longer. For this, we present a separate, simple proof for the Splitting lemma.

Recent related work on the combinatorial aspects of the satisfiability problem (see Kullmann [86] for a recent survey) makes use of the connection with partitioning complete graphs into complete bipartite graphs (bicliques). This connection, and in particular, the Graham–Pollak theorem [54] is used by Laborde [88] to show that a maximal k-term-DNF contains at least $2k-1$ variables. (This result, in turn, follows immediately from Theorem 8.1 above without using the Graham–Pollak theorem.) Section 8.5 contains an application of the Splitting Lemma (Lemma 8.10) showing that the family of recursive partitions into complete bipartite graphs has an extremal property among all partitions into complete bipartite graphs.

8.2 Further Definitions and Notations

The DNF φ is a **minimal cover** of the term t, if φ is a cover of t (i.e., t is an implicant of φ), but every DNF obtained from φ by deleting a term is not a cover of t.

Let t be a term, and $\varphi = t_1 \vee \cdots \vee t_k$ be a DNF. Every term t_i of φ can be uniquely written in the form

$$t_i = t'_i \wedge t''_i, \tag{8.1}$$

where t'_i contains all the literals from t_i which also occur in t, and t''_i contains the remaining literals of t_i.

Recall that for a DNF φ, $\mathrm{Var}(\varphi)$ (resp., $\mathrm{Lit}(\varphi)$) denotes the set of variables (resp., literals) occurring in any term of φ. Let

$$\mathrm{UnateLit}(\varphi) = \{u \in \mathrm{Lit}(\varphi) : \overline{u} \notin \mathrm{Lit}(\varphi)\} \tag{8.2}$$

be the set of **unate** literals in φ, i.e. the set of those literals occurring in φ, for which their negation does not occur in φ.

The **graph** of the n-dimensional cube has $\mathcal{A}_n$ as vertices, and edges $(\mathbf{x}, \mathbf{y})$ for every $\mathbf{x}, \mathbf{y} \in \mathcal{A}_n$ of Hamming distance 1. The distance of two subcubes Q_1 and Q_2 is $\min\{\mathrm{distH}(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in Q_1, \mathbf{y} \in Q_2\}$. Note that the distance of $\mathcal{T}(t_1)$ and $\mathcal{T}(t_2)$ is equal to the number of conflicts between the terms t_1 and t_2. A partition of the cube into subcubes can also be viewed as a disjoint DNF tautology. A partition of a cube into subcubes is **pairwise neighboring**, if any two subcubes in the partition have distance 1. A set of terms forms a pairwise neighboring partition if the corresponding set of cubes forms a pairwise neighboring partition.

8.3 Previous Results on k-term-DNFs and Prime Implicants

In this section we describe the results of [31; 88; 90; 97] on prime implicants of k-term-DNF. We give a complete, self-contained presentation in order to clarify what are the consequences of the separate assumptions of being an implicant, a prime implicant, resp. a minimal cover, and to give an explicit formulation of results implicit in [88]. We use the notation introduced above in (8.1) and (8.2).

Proposition 8.2 *A term t is an implicant of a DNF φ if and only if $\bigvee_{i=1}^k t_i'' = 1$.*

Proof
For the "if" direction, let $\mathbf{x}$ be a truth assignment such that $t(\mathbf{x}) = 1$. Then $t_i'(\mathbf{x}) = 1$ for every i and $t_i''(\mathbf{x}) = 1$ for some i, so $t_i(\mathbf{x}) = 1$ for some i, and thus $\varphi(\mathbf{x}) = 1$.

For the "only if" direction assume $\bigvee_{i=1}^k t_i'' \not\equiv 1$, i.e., $\left(\bigvee_{i=1}^k t_i''\right)(\mathbf{x}) = 0$ for some $\mathbf{x}$. The literals occurring in $\bigvee_{i=1}^k t_i''$ do not occur in t, but it may be the case that the negation of such a literal occurs in t. Let $\mathbf{y}$ be the truth assignment obtained from $\mathbf{x}$ by setting all the literals of t to 1. Then every literal in $\bigvee_{i=1}^k t_i''$ is either unchanged, or is changed to 0, thus $\left(\bigvee_{i=1}^k t_i''\right)(\mathbf{y}) = 0$, and so $\varphi(\mathbf{y}) = 0$. But $t(\mathbf{y}) = 1$, contradicting the fact that t is an implicant of φ. □

Proposition 8.3 *If t is a prime implicant of φ then*

(a) $t = \bigwedge_{i=1}^k t_i'$,

(b) *every literal of t occurs in φ.*

Proof
For (a), it follows from the definition that $t \leq \bigwedge_{i=1}^k t_i'$. Assume that a variable v in t does not occur in any t_i. Then v does not occur in φ at all, though $\overline{v}$ may occur in some t_i''. But then t is an implicant of the disjunction of those terms in φ which do not contain $\overline{v}$, and so by deleting v from t we still get an implicant of φ. Part (b) follows trivially from (a). □

Proposition 8.4 *If φ is a minimal cover of t then*

(a) $\mathrm{Lit}(t) \cap \mathrm{Lit}(\varphi) = \mathrm{UnateLit}(\varphi)$,

(b) $\bigvee_{i=1}^k t_i''$ *is a minimal cover of* 1.

Proof
To see that $\mathrm{Lit}(t) \cap \mathrm{Lit}(\varphi) \subseteq \mathrm{UnateLit}(\varphi)$ note that if t contains a non-unate literal ε of φ, then terms containing $\overline{\varepsilon}$ can be deleted from φ and we still get a cover of t, contradicting the minimality of φ. For the other direction of (a), assume that a unate literal ε is not contained in t. Then $\overline{\varepsilon}\, t$ is also an implicant of φ, which is covered by the terms of φ not containing ε. As these terms do not contain $\overline{\varepsilon}$ either, their disjunction covers t as well, again contradicting the minimality of φ. Part (b) follows from Proposition 8.2. □

Putting together Propositions 8.2, 8.3 and 8.4, we get the following.

Theorem 8.5 *If t is a prime implicant of φ and φ is a minimal cover of t, then*

(a) *t is the conjunction of the literals in $\mathrm{UnateLit}(\varphi)$,*

(b) *$\bigvee_{i=1}^{k} t_i''$ is a minimal cover of 1.*

Theorem 8.6 ([31; 90; 97]) *Every k-*term-DNF *has at most $2^k - 1$ prime implicants.*

Proof
Let φ be a k-term-DNF and t be a prime implicant of φ. Consider a minimal set of terms of φ covering t. Then, by Theorem 8.5 (a), t is uniquely determined by this nonempty set of terms. □

The next result gives important structural information on maximal DNF's.

Theorem 8.7 ([88]) *Let $\varphi = t_1 \vee \cdots \vee t_k$ be a k-*term-DNF *with $2^k - 1$ prime implicants, and let t be the term formed by the literals in $\mathrm{UnateLit}(\varphi)$. Then*

(a) *$\bigvee_{i=1}^{k} t_i''$ is a minimal cover of 1,*

(b) *t_i'' and t_j'' conflict in exactly one variable, for every $1 \leq i < j \leq k$.*

Proof
By Theorems 8.5 and 8.6, every nonempty subset of the terms of φ is a minimal covering of some prime implicant of φ. Part (a) follows by applying Theorem 8.5 (b) to all the terms.

Let us consider now $\varphi_{i,j} = t_i \vee t_j$. Again, this is a minimal cover of a prime implicant of φ. If t_i and t_j do not conflict in any variable, then, by Theorem 8.5 (a), the corresponding prime implicant is the term formed by all the literals in t_i and t_j. But that term is not a prime implicant. Indeed, it must be the case that $t_i \neq t_j$, and so $t_i \wedge t_j < t_i$ or $t_i \wedge t_j < t_j$. If t_i and t_j conflict in more than one variable, then we get a contradiction to Theorem 8.5 (b), as the disjunction of two terms with at least two conflicts cannot be 1. □

8.4 Proof of Theorem 8.1

In this section we prove Theorem 8.1: A DNF is maximal if and only if it corresponds to a NUD.

First we consider the "if" direction.

Lemma 8.8 *Every NUD corresponds to a maximal DNF.*

Proof
Let T be a k-NUD, and let H be a nonempty subset of its leaves. Define the term

$$t_H := \bigwedge \mathrm{UnateLit}(\{t_\ell : \ell \in H\}).$$

Let $\mathbf{x}$ be a truth assignment satisfying t_H. It follows by induction on the number of inner nodes evaluated, that on input $\mathbf{x}$ we arrive at a leaf belonging to H, and it follows from the definition of t_H that $\mathbf{x}$ satisfies every literal assigned to that leaf. Thus t_H is an implicant of φ_T.

Assume that we delete a v literal, say $\varepsilon = v_i$ from t_H, to get the term t'. (The $\varepsilon = \overline{v_i}$ case is symmetric.) As $\varepsilon \in \mathrm{UnateLit}(\{t_\ell : \ell \in H\})$, there is a leaf ℓ_1 belonging to H below the right child of the inner node labelled v_i, but no leaf below the left child of the node is in H. Let $\mathbf{x}$ be the assignment satisfying all the literals in t_{ℓ_1} and t_H, with those w literals that don't occur in these terms set to 0. Let $\mathbf{y} = \mathbf{x}^{[v_i]}$. On the input $\mathbf{y}$ we arrive at a leaf ℓ_2 below the left child of v_i. But the w literal assigned to ℓ_2 is set to 0 in $\mathbf{y}$, and hence $\varphi_T(\mathbf{y}) = 0$. On the other hand, $\mathbf{y}$ still satisfies t'. Thus t' is not an implicant.

Assume now that we delete a w literal, say $\varepsilon = w_j$, from t_H, to get the term t'. (The $\varepsilon = \overline{w_j}$ case is symmetric.) Let ℓ be the leaf containing ε. It follows from the definition of t_H that $\ell \in H$. Let $\mathbf{x}$ be an assignment satisfying t_ℓ and t_H, and let $\mathbf{y} = \mathbf{x}^{[w_j]}$. Then the input $\mathbf{y}$ leads to ℓ, but as the literal ε has value 0 for assignment $\mathbf{y}$, we get $\varphi_T(\mathbf{y}) = 0$. On the other hand, $\mathbf{y}$ still satisfies t'. Thus t' is not an implicant.

The case when we delete a u literal, say $\delta = u_j$ or $\delta = \overline{u_j}$, from t_H is the same, except now there may be several leaves in H containing δ. We can choose any such leaf, and repeat the previous argument. It again follows that the term obtained after deleting the literal is not an implicant.

Thus the term t_H is a prime implicant of φ_T. Terms corresponding to different subsets of L are different, as each leaf has its unique w literal. Hence φ_T has at least $2^k - 1$ prime implicants, and so it is maximal by Theorem 8.6. □

The rest of this section contains the proof of the "only if" direction of Theorem 8.1.

Lemma 8.9 *Every maximal DNF corresponds to a NUD.*

Proof
Let $\varphi = t_1 \vee \cdots \vee t_k$ be a k-term-DNF with $2^k - 1$ prime implicants. Consider the term $t = \mathrm{UnateLit}(\varphi)$, and the decomposition $t_i = t_i' \wedge t_i''$ of the terms of φ with respect to t, as in (8.1). According to Theorem 8.7, the terms $t_1'', \ldots, t_k''$ form a pairwise neighboring partition over the non-unate variables occurring in φ, i.e., over the s-dimensional cube, $\mathcal{A}_s$, where $s = |\mathrm{Var}(\varphi)| - |\mathrm{UnateLit}(\varphi)|$. The following lemma states a basic combinatorial property of pairwise neighboring partitions.

Lemma 8.10 (Splitting Lemma) *If a set of $k \geq 2$ terms forms a pairwise neighboring partition, then there is a variable that occurs (unnegated or negated) in every term.*

Proof

We proceed by induction on the number of variables; the case of one or two variables is trivial. Let $\hat{t}_1, \ldots, \hat{t}_k$ be terms forming a pairwise neighboring partition of the s-dimensional cube $\mathcal{A}_s$.

Consider the ε half cube corresponding to an arbitrary literal ε. The restriction of $\hat{t}_1, \ldots, \hat{t}_k$ to the ε half cube is formed by deleting terms which contain the literal $\overline{\varepsilon}$. It follows directly from the definitions that the restriction gives a pairwise neighboring partition of the ε half cube. If the restriction consists of a single cube then ε is a term of the original partition. In this case every other term of the original partition must contain $\overline{\varepsilon}$ and we are done. Hence in what follows we may assume that the restrictions always contain at least two terms.

Applying the induction hypothesis to the pairwise neighboring partition of the $s-1$ dimensional cube obtained by deleting the component corresponding to ε, and deleting the literal ε from each of the remaining terms, it follows that there is a variable $\mathrm{Split}(\varepsilon)$, different from the variable of ε, contained (negated or unnegated) in every term covering a point in the ε half cube. As there are $2s$ literals and s variables, there are literals ε_1 and ε_2 such that $\mathrm{Split}(\varepsilon_1) = \mathrm{Split}(\varepsilon_2) = u$ for some variable u.

We claim that u occurs (negated or unnegated) in every term of the partition $\hat{t}_1, \ldots, \hat{t}_k$. If ε_1 is the negation of ε_2, then u must occur in every term and we are done; henceforth we can assume that ε_1 and ε_2 have different variables. Assume now for contradiction that u is not in every term of the partition. Let $\tilde{t}$ be a term of the partition containing neither u nor $\overline{u}$, and let $\mathbf{x}$ be a point in $\mathcal{T}\left(\tilde{t}\right)$. Then $\mathbf{x}$ belongs to neither the ε_1 subcube, nor the ε_2 subcube.

Consider the points $\mathbf{x}^{[\varepsilon_1]}$ and $\mathbf{x}^{[\varepsilon_2]}$, covered respectively by terms $\tilde{t}_{\varepsilon_1}$ and $\tilde{t}_{\varepsilon_2}$ of the partition. Note that $\tilde{t}_{\varepsilon_1}$ and $\tilde{t}_{\varepsilon_2}$ are different. Indeed, if $\tilde{t}_{\varepsilon_1} = \tilde{t}_{\varepsilon_2}$ then, as $\mathbf{x}^{[\varepsilon_1]}$ and $\mathbf{x}^{[\varepsilon_2]}$ differ in both their ε_1 and ε_2 components, $\tilde{t}_{\varepsilon_1}$ (and thus $\tilde{t}_{\varepsilon_2}$) contains neither ε_1 nor ε_2, and hence it covers $\mathbf{x}$ as well. This contradicts the definition of $\mathbf{x}$.

The points $\mathbf{x}^{[\varepsilon_1]}$ and $\mathbf{x}^{[\varepsilon_2]}$ differ only in their ε_1 and ε_2 components; hence the unique conflict of the terms $\tilde{t}_{\varepsilon_1}$ and $\tilde{t}_{\varepsilon_2}$ is either ε_1 or ε_2. Assume without loss of generality that the conflict is ε_1, and that $\tilde{t}_{\varepsilon_1}$ contains ε_1 and $\tilde{t}_{\varepsilon_2}$ contains $\overline{\varepsilon_1}$. By definition, both $\tilde{t}_{\varepsilon_1}$ and $\tilde{t}_{\varepsilon_2}$ contain either u or $\overline{u}$. As $\mathbf{x}^{[\varepsilon_1]}$ and $\mathbf{x}^{[\varepsilon_2]}$ do not conflict on u, both $\tilde{t}_{\varepsilon_1}$ and $\tilde{t}_{\varepsilon_2}$ must contain variable u with the same orientation; say u appears unnegated in both. Thus so far we have that $\varepsilon_1, u \in \mathrm{Lit}\left(\tilde{t}_{\varepsilon_1}\right)$ and that

$$\overline{\varepsilon}, u \in \mathrm{Lit}\left(\tilde{t}_{\varepsilon_2}\right).$$

Now consider the point $\mathbf{x}^{[\varepsilon_1,u]}$ covered by the term $\tilde{t}_{\varepsilon_1,u}$ of the partition. As $\mathbf{x}^{[\varepsilon_1,u]}$ is in the ε_1 subcube, it contains either u or $\overline{u}$; but as $\mathbf{x}^{[\varepsilon_1,u]}(u) = 0$, it must be $\overline{u}$. What is the unique conflict of $\tilde{t}$ (the term covering $\mathbf{x}$) and $\tilde{t}_{\varepsilon_1,u}$? As $\mathbf{x}^{[\varepsilon_1,u]}$ and $\mathbf{x}$ conflict only on their ε_1 and u components, but $\tilde{t}$ contains neither u nor $\overline{u}$, thus it must be ε_1. Then

$$\varepsilon_1, \overline{u} \in \mathrm{Lit}\left(\tilde{t}_{\varepsilon_1,u}\right),$$

which means that $\tilde{t}_{\varepsilon_2}$ and $\tilde{t}_{\varepsilon_1,u}$ conflict in at least two components, a contradiction. □

The Splitting Lemma is now used to prove the characterization of nonrepeating decision tree tautologies mentioned in the introduction.

Lemma 8.11 (ND Lemma [85]) *A set of $k \geq 2$ terms forms a pairwise neighboring partition if and only if it is an ND.*

Proof

Apply Lemma 8.10 to the pairwise neighboring partition to get a variable v_1 occurring in every term. It must be the case that v_1 occurs both unnegated and negated, as otherwise the cubes would not cover the whole cube. If the $\mathcal{T}(v_1)$ (resp. the $\mathcal{T}(\overline{v_1})$) half cube contains just one cube then we stop at that branch, otherwise we use the lemma again to get a variable which occurs in every subcube of the partition, belonging to the $\mathcal{T}(v_1)$ (resp. $\mathcal{T}(\overline{v_1})$) half cube, etc. In this way we get a tree, where the inner nodes are labeled with variables and there are k leaves $\ell_1, \ldots, \ell_k$ corresponding to the cubes in the partition. (The tree constructed is (the dual of) a special *search tree* in the sense of [93] for the partition.) The labels of the inner nodes are *different*, as the same label appearing twice would mean that some pair of cubes have distance at least 2. Indeed, if variable v_i occurs twice then let v_j be the variable labeling the least common ancestor of the two occurrences in the tree. By construction, there are terms containing $\overline{v_i}\,\overline{v_j}$, resp. $v_i\, v_j$. Thus the partition is an ND. □

Now we can complete the proof of Lemma 8.9. Lemma 8.11 gives a nonrepeating decision tree for the pairwise neighboring terms $t''_1, \ldots, t''_k$. We claim that by adding the literals in t'_i to the leaf ℓ_i, we get a k-NUD for φ. Consider any truth assignment $\mathbf{x}$ to the variables in φ. Evaluating the tree on $\mathbf{x}$, we arrive at a leaf corresponding to a term t''_i. As $\varphi(\mathbf{x}) = 1$ iff $t'_i(\mathbf{x}) = 1$, the tree computes φ correctly. By construction, all the literals in the leaves are unate. Thus, in order to verify the NUD-ity of the tree, it only remains to show that for every leaf there is a literal which occurs only in that leaf (that literal will be its w literal). Assume that this is not the case, and every (unate) literal assigned to leaf ℓ_i occurs in some other leaf. Let ε be the last literal on the path leading to ℓ_i. Then $\overline{\varepsilon} \in \mathrm{UnateLit}(\varphi \setminus \{t_i\})$. We claim that $\mathrm{UnateLit}(\varphi \setminus \{t_i\}) \setminus \{\overline{\varepsilon}\}$ is an implicant of φ. Let $\mathbf{x}$ be a truth assignment satisfying every literal in $\mathrm{UnateLit}(\varphi \setminus \{t_i\}) \setminus \{\overline{\varepsilon}\}$, and let us evaluate the tree on $\mathbf{x}$. If we arrive at a leaf other than ℓ_i, then $\varphi(\mathbf{x}) = 1$ by construction. But $\varphi(\mathbf{x}) = 1$ if we arrive at ℓ_i as well, as all unate literals in ℓ_i occur in other leaves, and thus they must be set to 1 in $\mathbf{x}$. Thus $\mathrm{UnateLit}(\varphi \setminus \{t_i\})$ is not a prime implicant of φ, contradicting Theorems 8.5 and 8.6. □

8.5 A Graph Theoretic Application of the Splitting Lemma

Given a set of pairwise disjoint cubes in the n-dimensional cube $\mathcal{A}_n$, corresponding to terms $t_1, \ldots, t_k$, one can construct a covering

$$\mathcal{G} = \{G_1, \ldots, G_n\}$$

of the k-vertex complete graph K_k by complete bipartite graphs, where G_r has an edge connecting vertices x_i and x_j if terms t_i and t_j conflict in the variable v_r. If the set of cubes is pairwise neighboring, then this covering is a partition, as the complete bipartite graphs are edge disjoint.

Conversely, given a covering $\mathcal{G} = \{G_1, \dots, G_n\}$ of K_k by complete bipartite graphs, we can construct a set of pairwise disjoint cubes $t_1, \dots, t_k$ in $\{0,1\}^n$. For every G_r fix arbitrarily one of the sides as the left side. The term t_i contains v_r (resp. $\overline{v_r}$), if vertex x_i is contained in the left (resp. right) side of G_r. If $\mathcal{G}$ is a partition, then it follows that the t_i's are pairwise neighboring. The cubes thus constructed do not necessarily form a partition of $\mathcal{A}_n$ (an example is given below).

The Graham–Pollak theorem [54] states that every partition of K_k into complete bipartite graphs consists of at least $k-1$ graphs. A large class of such partitions, which can be called **recursive** partitions, is obtained as follows. Take a complete bipartite graph on the whole vertex set. This 'takes care' of all edges connecting the two sides. In order to partition the remaining edges (those having both endpoints in the same side), repeat the same construction, i.e., recursively add similar partitions of the complete graphs formed by the two sides of this bipartite graph (see, e.g., [19]).

Consider a partition $\mathcal{G} = \{G_1, \dots, G_n\}$ of K_k into complete bipartite graphs. Let the degree of a vertex x with respect to $\mathcal{G}$, denoted by $d_{\mathcal{G}}(x)$, be the number of G_i's containing x, and let the **volume** $\mathrm{Vol}(\mathcal{G})$ of the partition be defined as

$$\mathrm{Vol}(\mathcal{G}) = \sum_x 2^{-d_{\mathcal{G}}(x)}.$$

In view of the translation into a set of pairwise disjoint cubes in $\mathcal{A}_n$ described above, $\mathrm{Vol}(\mathcal{G}) \le 1$ for every $\mathcal{G}$, as $d_{\mathcal{G}}(x_i) = |t_i|$ for every $i = 1, \dots, k$, and $\mathrm{Vol}(\mathcal{G}) = 1$ if and only if the cubes form a partition of $\mathcal{A}_n$. For example, the partition of K_4 into the 3 complete bipartite graphs $(\{1\},\{3,4\})$, $(\{2\},\{1,4\})$, and $(\{3\},\{2,4\})$ (mentioned in [88]) has volume $\frac{7}{8}$. This partition of K_4 is not recursive. (It was actually this example which suggested Lemma 8.10.) As a corollary to the Splitting Lemma (Lemma 8.10) one gets the following characterization of recursive partitions. This characterization is also a direct consequence of Kullmann's [84–86] results.

Corollary 8.12 *A partition $\mathcal{G}$ is recursive if and only if* $\mathrm{Vol}(\mathcal{G}) = 1$.

Proof

The "only if" direction follows directly by induction on the number of vertices by considering the bipartite graph from $\mathcal{G}$ which contains all the vertices.

For the "if" direction, one only has to note that the set of terms $t_1, \dots, t_k$ constructed above is pairwise neighboring, and by the volume condition it is also a partition of the whole cube.

Applying Lemma 8.10 we get that there is a variable which occurs (unnegated or negated) in every term. This means that the corresponding bipartite graph contains all the k vertices. The remaining partitions of the two sides of this bipartite graph have total volume 2, and thus each side must have volume 1. The statement then follows by induction. □

The corollary shows that among partitions of K_k into complete bipartite graphs, recursive ones have the largest possible volume. Among the partitions of K_k into $k-1$ complete bipartite graphs, which ones have *minimal* volume?

8.6 Concluding Remarks

In this chapter k-term-DNF with the largest number of prime implicants were discussed. Similar results do not appear to be known for **shortest** prime implicants, i.e., prime implicants containing the smallest possible number of literals. The k-term-DNF

$$v_1\overline{v_2} \vee v_2\overline{v_3} \vee \cdots \vee v_{k-1}\overline{v_k} \vee v_k\overline{v_1},$$

which is false for $\mathbf{0}$ and $\mathbf{1}$, and true everywhere else, has $k(k-1)$ prime implicants, namely $v_i\overline{v_j}$ for every $i \neq j$. These prime implicants are all shortest prime implicants, as the DNF has no prime implicants consisting of a single literal. How many shortest prime implicants can a k-term-DNF have in general?

Another question concerns the maximal number of prime implicants of a Boolean function which is true at a given number of points. As noted by Levin [90], every implicant is determined by the top and bottom of the corresponding subcube, in the componentwise partial ordering of the hypercube (the top and bottom may also be identical). Thus if a function is true at m points, then it has $O(m^2)$ prime implicants. It is also noted in [90] that the n-variable function which is true for assignments of weight between $\frac{n}{3}$ and $\frac{2n}{3}$, has $m^{\log 3 - o(1)}$ prime implicants. (This is the function with the largest known number of prime implicants among n-variable functions.) Thus the maximal number of prime implicants is bounded by two polynomial functions of m, and the question is to get sharper bounds.

Finally note that the results presented in this chapter—unless noted otherwise (like in the case of the results dicussed in Section 8.3)—appeared in the paper [114], co-authored by the author of the present dissertation.

Chapter 9

Disjoint DNF Tautologies with Conflict Bound Two

One of the main ingredients in the proof of the characterization result in the previous chapter was the ND Lemma (Lemma 8.11), which can be formulated both using the

- syntactic wiew: that the class of DDNF tautologies with conflict bound one (i.e., DNFs with terms conflicting in one variable pairwise) are NDs (i.e., DNFs generated by labeled binary trees with each inner node having a unique label), and using the

- semantic view: that in every pairwise neighbouring partition of the n-dimensional cube there is a **perfect split**: a split of the cube in two complementary half cubes such that each subcube of the partition is contained in either one of the half cubes.

These two views offer two effectively different directions for further investigations; these directions are discussed in the next section. However, somewhat surprisingly, for one more step these directions do not separate. More precisely, we shall see in this chapter that the following strengthening of the ND Lemma holds: any DDNF tautology with conflict bound two can also be generated by some labeled binary trees—or, equivalently, for any cube partition with pairwise distance bounded by two there is a perfect split similar as above.

Throughout the notations and terminology introduced in the previous chapter are used.

9.1 Characterization of DDNF tautologies with Conflict Bound Two

This section discusses both of the two different directions mentioned above. More precisely:

- the direction suggested by syntax, considering DDNF and LBT generated tautologies
- the direction suggested by semantics, considering the general splitting problem for cube partitions,

—furthermore how the strengthening of the ND Lemma gets realized in these two settings.

9.1.1 Syntactic View: DDNF tautologies and LBT generated DNFs

A decision tree (and, of course, also an LBT) naturally encodes a DNF tautology consisting of the terms corresponding to the leaves of the tree, where the term corresponding to a leaf consists of the literals labelling the edges on the path from the root to the leaf. These DNF tautologies hold the following special properties:

(a) the terms are pairwise conflicting, and

(b) the terms possess a hierarchical structure: there is a variable v that appears in each of them; there is a variable w that appears in every term containing literal v and there is a variable u that appears in every term containing literal $\overline{v}$ (w and u may be identical); and so on.

Such DNFs are called **binary tree generated DNFs**, or BT-DNFs for short (for a formal definition see Section 9.2); recall on the other hand that DNFs possessing property (a) but not necessarily property (b) are called **disjoint DNFs**, or DDNFs. The question thus naturally arises, how special do these properties make a decision tree, regarding complexity. This question was investigated by Lovász *et al.* in [93]. More precisely they were interested in the following problem: given a DNF tautology φ, the task is to construct a decision tree such that for each term of the DNF generated by it there is a term of φ that is a subterm of it. They have shown that for some very "small" DNF tautologies this problem can be solved only with "extremely large" decision trees [1].

On the other hand, the ND Lemma (Lemma 8.11) states that, when restricting the DNFs to the subclass posessing property (a) (i.e., the class of DDNFs), *and* further bounding the number of conflicts between the terms to one (i.e., for each pair of terms there is *exactly* one variable appearing negated in one of them and unnegated in the other), then the resulting class consists of DNFs that can all be generated by decision trees.

In this chapter we give a strengthening of the above result, showing that the conflict bound can be relaxed to two:

[1]They measure the complexity by the **depth** of the DNF (resp. decision tree), which is the maximal number of literals appearing in a term of the given DNF (resp. of the BT-DNF generated by the tree). What they show is that for some constant depth DNFs one needs decision trees of depth linear (thus maximal) in the number of variables.

Theorem 9.1 *If φ is a DDNF tautology with terms conflicting in one or two variables pairwise, then φ is a* BT-DNF.

Example 9.1
The DNF

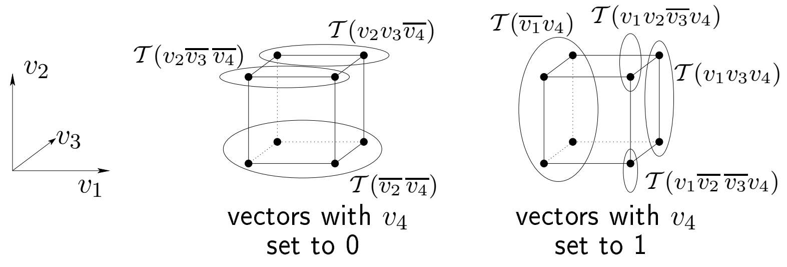
$$\varphi_{\text{ex9.1}} = \overline{v_2}\,\overline{v_4} \vee v_2\overline{v_3}\,\overline{v_4} \vee v_2v_3\overline{v_4} \vee \overline{v_1}v_4 \vee v_1\overline{v_2}\,\overline{v_3}v_4 \vee v_1v_2\overline{v_3}v_4 \vee v_1v_3v_4$$

is a DDNF with conflict bound two, and Figure 9.2 proves that it is also a BT-DNF— which is also apparent writing $\varphi_{\text{ex9.1}}$ in the form

$$\varphi_{\text{ex9.1}} = \overline{v_4}\,\overline{v_2} \vee \overline{v_4}v_2\overline{v_3} \vee \overline{v_4}v_2v_3 \vee v_4\overline{v_1} \vee v_4v_1\overline{v_3}\,\overline{v_2} \vee v_4v_1\overline{v_3}v_2 \vee v_4v_1v_3,$$

or also from Figure 9.1, visualizing the relations of the truth sets of the various terms.

Figure 9.1: The assignments to variables v_1, v_2, v_3 and v_4 represented as the vertices of the 4-dimensional hypercube and grouped according to which term of $\varphi_{\text{ex9.1}}$ they satisfy.


v_2
v_3
v_1
$\mathcal{T}(v_2\overline{v_3}\,\overline{v_4})$
$\mathcal{T}(v_2v_3\overline{v_4})$
$\mathcal{T}(\overline{v_2}\,\overline{v_4})$
vectors with v_4 set to 0
$\mathcal{T}(\overline{v_1}v_4)$
$\mathcal{T}(v_1v_2\overline{v_3}v_4)$
$\mathcal{T}(v_1v_3v_4)$
$\mathcal{T}(v_1\overline{v_2}\,\overline{v_3}v_4)$
vectors with v_4 set to 1

Note however that the result of Theorem 9.1 does not generalize to conflict bound three, as the following example demonstrates.

Example 9.2
DDNF $\varphi_{\text{ex9.2}} = v_1v_3 \vee \overline{v_1}v_2 \vee \overline{v_2}\,\overline{v_3} \vee \overline{v_1}\,\overline{v_2}v_3 \vee v_1v_2\overline{v_3}$ is a tautology and has terms conflicting in at most three variables pairwise, but is not a BT-DNF. (Simply note that there is no variable that appears in every term.)

Note also that heorem 9.1 implies the following characterization result.

Corollary 9.2 *φ is a DDNF tautology with conflict bound two if and only if φ is a* BT-DNF *with conflict bound two.*

Finally we mention that a related problem is the problem of representing a Boolean function f as a DNF or as a decision tree—that is, when one needs to construct a DNF tautology (resp. decision tree) with each term (resp. with each term of the corresponding BT-DNF) covering only assignments that satisfy f, or only assignments that falsify f—, and one is interested in comparing the complexity of the two class in this setting. See for example [73; 110; 121].

9.1.2 Semantic View: The General Splitting Problem for Cube Partitions

According to the Splitting Lemma (Lemma 8.10), for every pairwise neighboring cube partition, the whole cube can be split into two halves in such a way that every cube of the partition is contained in one of the halves. The following question thus rises naturally: what can be said without the pairwise neighboring property? Given an arbitrary partition of the whole cube into subcubes and a split into two halves, let us say that a cube in the partition is **uncut**, if it is contained in either one of the halves. We would like to find a split such that the uncut cubes contain many points.

Thus we consider the following quantities. Given a cube partition φ over the variables $v_1, \ldots, v_n$ and a variable v_j, let

$$\nu_{\varphi,j} = \sum \left\{2^{-|t|} : t \in \varphi,\ v_j \in t \text{ or } \overline{v_j} \in t\right\}$$

be the fraction of the volume of uncut cubes in φ with respect to the v_j split of the cube, and let

$$\alpha_n = \min_{\varphi} \max_{1 \le j \le n} \nu_{\varphi,j},$$

where φ ranges over all cube partitions, or in other words, over all disjoint DNF tautologies. Note that as φ is a partition it holds that

$$\sum_{t \in \varphi} 2^{-|t|} = 1. \tag{9.1}$$

Theorem 9.3

$$\frac{\log n - \log\log n}{n} \le \alpha_n \le O\left(n^{-\frac{1}{5}}\right).$$

Proof
Let $\varphi = t_1 \vee \cdots \vee t_r$ be a disjoint DNF tautology over the variables $v_1, \ldots, v_n$. If the term t_i contains v_j or $\overline{v_j}$, then t_i contributes $2^{-|t_i|}$ to $\nu_{\varphi,j}$. Thus

$$\sum_{j=1}^{n} \nu_{\varphi,j} = \sum_{i=1}^{r} |t_i| \cdot 2^{-|t_i|},$$

and there is a variable v_j with

$$\nu_{\varphi,j} \ge \frac{1}{n} \sum_{i=1}^{r} |t_i| \cdot 2^{-|t_i|}.$$

Let s denote the size of the shortest term in φ. As every term has size at least s, it follows from (9.1) that

$$\frac{1}{n}\sum_{i=1}^{r}|t_i|\cdot 2^{-|t_i|} \geq \frac{s}{n}\sum_{i=1}^{r}2^{-|t_i|} = \frac{s}{n}.$$

On the other hand, for every variable v_j occurring in a shortest term t_i it holds that $\nu_{\varphi,j} \geq 2^{-s}$. Thus

$$\alpha_n \geq \min\left(\frac{s}{n}, 2^{-s}\right). \tag{9.2}$$

The lower bound then follows by taking $s = \log n - \log\log n$, for which the two terms in (9.2) are close to each other.

The upper bound follows from a construction of Savický and Sgall [111], providing an upper bound on the number of variable occurrences in tautological k-DNF formulas (a problem introduced by Tovey [122] and Kratochvíl, Savický and Tuza [83]). They constructed disjoint DNF tautologies over $n = 4^\ell$ variables, having 2^{3^ℓ} terms of size 3^ℓ, such that every variable occurs in at most a

$$\left(\frac{3}{4}\right)^\ell$$

fraction of the terms. The bound then follows by a direct calculation. □

We note that the upper bound of Savický and Sgall [111] has recently been improved almost optimally by Hoory and Szeider [70]. The improved constructions do not appear to improve the bound above, since the DNF constructed are not disjoint.

Already Theorems 8.1 and 9.3 suggest that it may be of interest to consider the quantity α_n^d, which is defined as α_n, except that φ is restricted to cube partitions with pairwise distances bounded by d. (For example in the construction of [111] the maximal distance grows linearly with n.) The main result presented in this chapter is that $\alpha_n^2 = 1$ (for any positive integer n); but note also that this does not generalize to $d = 3$: Example 9.2 proves that $\alpha_3^3 < 1$.

9.2 Further Definitions and Notations

In an LBT a path from the root to a leaf naturally determines a term obtained by simply conjuncting the literals appearing in the labels of the edges along the path. Thus, given a decision tree, the terms corresponding to its leaves put up a DDNF tautology [2]. Recall that such DDNF tautologies are called **binary tree generated DNF**s, or **BT-DNF**s for short. Alternatively, one can define the class of BT-DNFs as the smallest subset DT-DNF of the set of DNFs satisfying:

- If x is a variable, then the DNF $x \vee \overline{x}$ is an element of DT-DNF.

[2] Note that in Chapter 8 non-repeating decision tree tautologies were constructed in the similar fashion using non-repeating unate-leaf decision trees.

- If x is a variable and both $T_1 \vee \cdots \vee T_k$ and $T'_1 \vee \cdots \vee T'_\ell$ are elements of DT-DNF, then the DNF $(x \wedge T_1) \vee \cdots \vee (x \wedge T_k) \vee (\overline{x} \wedge T'_1) \vee \cdots \vee (\overline{x} \wedge T'_\ell)$ is also an element of DT-DNF.

Note that in case φ is a DDNF tautology, then there is a *unique* term of φ satisfied by truth assignment $\mathbf{x}$; denote it $t_{\mathbf{x}}(\varphi)$. When it causes no ambiguity, φ is omitted and simply $t_{\mathbf{x}}$ is used instead.

9.3 Proof of Theorem 9.1

For simplicity assume that $\mathcal{V}'$ is the set of variables in focus.

Theorem 9.1 is proved by induction on the number of terms in φ. In case φ contains one or two terms, the statement is obvious. Now we show that φ is a BT-DNF, assuming:

Induction hypothesis: DDNF φ with conflict bound two contains $r \geq 3$ terms, and the statement holds for any DDNF tautology with conflict bound two having less than r terms. (9.3)

Let t be an arbitrary term of φ. Assume without loss of generality that $t = v_1 \cdots v_k$. Of course, if φ is a BT-DNF, then for some $1 \leq i \leq k$ φ has a subformula equivalent to $v_1 \cdots v_{i-1} v_{i+1} \cdots v_k$: namely the one induced by the parent node of the leaf corresponding to t. (For example if $\varphi = \varphi_{\text{ex9.1}}$ from Example 9.1 and $t = v_1 v_3 v_4$, then $i = 3$, and the subformula $v_1 \overline{v_2}\, \overline{v_3} v_4 \vee v_1 v_2 \overline{v_3} v_4 \vee v_1 v_3 v_4$ of φ is equivalent to $t \setminus \{v_i\} = v_1 v_4$.) The next claim considers the reverse of this implication. (Also, for an example demonstrating the claim see Example 9.3.)

Claim 9.4 *Assume* (9.3)*, and let* $t = v_1 \cdots v_k$ *be a term of* φ*. Suppose that for some* $i \in \{1, \ldots, k\}$ *it holds that every term in* φ *that conflicts with* t *only in* v_i *contains* $v_1 \cdots v_{i-1} v_{i+1} \cdots v_k$ *as a subterm. Then* φ *is a* BT-DNF.

Proof
Consider the following sets

$$\begin{aligned}
S_1 =& \{\mathbf{x} \in \{0,1\}^n : \mathbf{x}^{[v_i]} \in \mathcal{T}(t)\}, \\
S_2 =& \mathcal{T}(v_1 \cdots v_{i-1} \overline{v_i} v_{i+1} \cdots v_k), \\
S_3 =& \cup_{t' \in \varphi : v_1 \cdots v_{i-1} \overline{v_i} v_{i+1} \cdots v_k \text{ is a subterm of } t'} \mathcal{T}(t'), \\
S_4 =& \cup_{t' \in \varphi : t \otimes t' = \{v_i\}} \mathcal{T}(t').
\end{aligned}$$

Then $S_1 = S_2$ and $S_2 \supseteq S_3$ always hold, and $S_3 \supseteq S_4$ follows from the condition of the Claim. However $S_1 \subseteq S_4$ is also true because

- since φ is a tautology, each element $\mathbf{x}$ of S_1 appears in some $\mathcal{T}(t')$ for some $t' \in \varphi$—recall that this t' is the term we denote as $t_{\mathbf{x}}(\varphi)$—, and

- since φ is a DDNF, each of these $t_{\mathbf{x}}(\varphi)$ terms must conflict with t in some variable. But this variable must be v_i, and only v_i, as the first k bit of each $\mathbf{x} \in S_1$ is 1, *except for the i-th bit*.

Thus all of the above sets are identical. Then defining

$$\varphi_1 := \{t' \in \varphi : v_1 \cdots v_{i-1}\overline{v_i}v_{i+1} \cdots v_k \text{ is a subterm of } t'\}$$

and

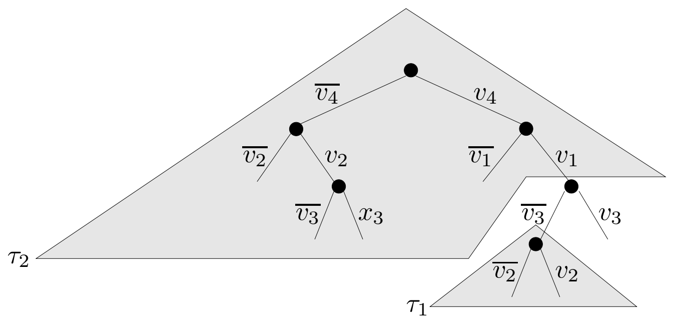
$$\varphi_2 :=(\varphi \setminus (\varphi_1 \cup \{t\})) \cup \{v_1 \cdots v_{i-1}v_{i+1} \cdots v_k\}$$

it holds that both $\varphi_1' := \{t' \setminus \{v_1, \cdots, v_{i-1}, \overline{v_i}, v_{i+1}, \cdots, v_k\} : t' \in \varphi_1\}$ and φ_2 are DDNF tautologies. Furthermore both have less terms then φ, thus by the induction hypothesis both are BT-DNFs. This immediately implies the Claim: pick an LBT τ_1 for φ_1' and an LBT τ_2 for φ_2, expand τ_1 to an LBT for $v_i \vee \{\overline{v_i} \wedge t' : t' \in \varphi_1'\}$ in the natural way, and paste it into τ_2 in the place of the leaf corresponding to $v_1 \cdots v_{i-1}v_{i+1} \cdots v_k$.
$\square$

Example 9.3

Demonstrating Claim 9.4, let $\varphi = \varphi_{\text{ex9.1}}$ from Example 9.1 and let $t = v_1v_3v_4$. Then $i = 3$, $\varphi_1 = v_1\overline{v_2}\,\overline{v_3}v_4 \vee v_1v_2\overline{v_3}v_4$, $\varphi_1' = \overline{v_2} \vee v_2$ and $\varphi_2 = (\overline{v_2}\,\overline{v_4} \vee v_2\overline{v_3}\,\overline{v_4} \vee v_2v_3\overline{v_4} \vee \overline{v_1}v_4) \vee v_1v_4$. See also Figure 9.2 for the decision tree τ_1 (resp. τ_2) for φ_1' (resp. φ_2).

Figure 9.2: Marking τ_1 and τ_2 on the decision tree generating $\varphi_{\text{ex9.1}}$ from Example 9.1. The labels of the nodes are omitted for simplicity.



Defining the following directed graph $G(V, E) = G_{\varphi,t}(V_{\varphi,t}, E_{\varphi,t})$:

$$\begin{aligned} V =& \{t' \in \varphi : |t \otimes t'| = 1 \text{ and } \mathrm{Var}(t') \not\supseteq \mathrm{Var}(t)\}, \\ E =& \{(t', t'') \in V^2 : \overline{v_i} \in t' \text{ and } v_i \notin \mathrm{Var}(t'') \text{ for some } 1 \le i \le k\}, \end{aligned} \tag{9.4}$$

based on Claim 9.4 one can give the following sufficient condition for φ being a BT-DNF (which, as one can easily show, is also a necessary condition):

Claim 9.5 *Assume* (9.3)*, let $t = v_1 \cdots v_k$ be a term of φ, and let $G = G_{F,T}$ be the graph defined as in* (9.4)*. If G contains no cycle, then φ is a* BT-DNF.

Proof

We show that if φ is not a BT-DNF, then G contains a cycle. Suppose thus that φ is not a BT-DNF. By Claim 9.4 this can only be if for $i = 1, \dots, k$ there is a term $t_i \in \varphi$ containing $\overline{v_i}$, containing no other variable from t negated, and having at least one of the variables in t missing. Consequently $t_1, \dots, t_k \in V$, and in the subgraph induced by them, each vertex has indegree at least one. The subgraph has thus no sink, implying that it contains a cycle. (For example if $\varphi = \varphi_{\text{ex9.2}}$ from Example 9.2 and $t = v_1v_3$, then V consists of the terms $t_1 = \overline{v_1}v_2$ and $t_2 = \overline{v_2}\,\overline{v_3}$, and there is an edge in E both from t_1 to t_2 and from t_2 to t_1—and thus G contains a cycle [3]: t_1, t_2, t_1.) □

In the rest of the paper we show that G indeed contains no cycle. Assume for the contradiction that this is not the case, and let $t_1, \dots, t_\ell, t_1$ be a cycle of minimal length (then of course $\ell \le k$), and assume without loss of generality that $\overline{v_i} \in t_i$, $i = 1, \dots, \ell$. (Note that no other variable of t appears unnegated in t_i, as $t_i \in V$.) Then for any distinct indices $i, j \in \{1, \dots, \ell\}$,

- if t_j follows t_i in the cycle [4], then $v_i \notin t_j$ (by the construction of E),
- if not, then $v_i \in t_j$, as otherwise $(t_i, t_j) \in E$, which would shortcut the cycle, and contradict that it is of minimal length.

These observations are summarized in Figure 9.3.

Figure 9.3: The cycle $t_1, \dots, t_\ell, t_1$. In the row of a term: "$+$" means that the given variable appears unnegated in it, "$-$" means that it appears negated in it, and " $\cdot$ " means that it does not appear in it. Consecutive elements of the cycle might conflict in other variables too, but non-consecutive elements have no more conflict.

	v_1	v_2	v_3	v_4	$\cdots$	$v_{\ell-2}$	$v_{\ell-1}$	v_ℓ
t	$+$	$+$	$+$	$+$	$\cdots$	$+$	$+$	$+$
t_1	$-$	$+$	$+$	$+$	$\cdots$	$+$	$+$	$\cdot$
t_2	$\cdot$	$-$	$+$	$+$	$\cdots$	$+$	$+$	$+$
t_3	$+$	$\cdot$	$-$	$+$	$\cdots$	$+$	$+$	$+$
t_4	$+$	$+$	$\cdot$	$-$	$\cdots$	$+$	$+$	$+$
$\vdots$					$\ddots$			
t_ℓ	$+$	$+$	$+$	$+$	$\cdots$	$+$	$\cdot$	$-$

Let us now investigate how these terms "behave" on the rest of the variables. The above observation obviously implies that if terms t_i and t_j are not consecutive elements of the cycle, then they do not conflict in variables $v_{\ell+1}, \dots, v_n$, as otherwise they would conflict in at least three variables: v_i, v_j and $v_{\ell'}$ for some $\ell \le \ell' \le n$. The question is, whether two consecutive elements of the cycle can (or have to) have some further conflicts. An equivalent (semantic) formulation of this question is whether there exists a (partial) assignment to variables $v_{\ell+1}, \dots, v_n$ consistent with the two terms. (Again, for an example demonstrating the claim see Example 9.4.)

[3]Which is in accordance with the fact that $\varphi_{\text{ex9.2}}$ is not a BT-DNF.

[4]That is, $j = i + 1$ if $i < \ell$, and $j = 1$ if $i = \ell$.

Lemma 9.6 *Assume* (9.3)*, let* $t = v_1 \cdots v_k$ *be a term of* φ *with* $k < n$*, let* $G = G_{\varphi,t}$ *defined as in* (9.4)*, and let* $t_1, \dots, t_\ell$ *be a cycle of minimal length in* G *as in Figure* 9.3*. Then there is no partial assignment for variables* $v_{\ell+1}, \dots, v_n$ *that is consistent with* t *and all of* $t_1, \dots, t_\ell$*.*

Proof
Suppose that t is of length less then n and assume for the contradiction that σ is a partial assignment for variables $v_{\ell+1}, \dots, v_n$ consistent with $t, t_1, t_2, \dots, t_\ell$. Let φ' be the DDNF consisting of the terms of φ that are consistent with σ, (thus t and $t_1, \dots, t_\ell$ are in φ'), and let φ'' be the DDNF tautology obtained from φ' by removing all occurrances of variables $v_{\ell+1}, \dots, v_n$. By the induction hypotheses φ'' is a BT-DNF [5], consequently for some $i \in \{1, \dots, \ell\}$ variable v_i occurs (negated or unnegated) in every term of φ'', and thus also in every term of φ'—in particular in each of $t_1, \dots, t_\ell$. But the term following t_i in the cycle contains neither v_i nor $\overline{v_i}$—a contradiction. (The condition $k < n$ is necessary since the partial assignment with empty domain is consistent with all terms.) □

Example 9.4
Let $\varphi = \varphi_{\text{ex9.1}}$ from Example 9.1, and let $t = v_1 v_3 v_4$. Then V contains terms $t_1 = \overline{v_1} v_4$ and $t_2 = v_2 v_3 \overline{v_4}$, and E contains the edge (t_1, t_2). As φ is a BT-DNF, by Lemma 9.6 (or, more precisely, by the proof of the lemma), some variable of t (i.e., one of v_1, v_3 and v_4) must occur in t_1 and t_2—and indeed: v_4 occurs unnegated in t_1 and negated in t_2.

The next lemma rules out another case: when there is exactly one pair of consecutive elements of the cycle that conflict in two variables.

Lemma 9.7 *Assume* (9.3)*, let* $t = v_1 \cdots v_k$ *be a term of* φ *with* $k < n$*, let* $G = G_{\varphi,t}$ *defined as in* (9.4)*, and let* ℓ *be the length of the smallest cycle in* G*. Unless* $\ell = 2$*, there is no cycle in* G *of length* ℓ *with the property that one pair of consecutive elements of the cycle conflict in two variables, and all other consecutive pairs conflict in one.*

Proof
Assume for the contradiction that $t_1, \dots, t_\ell, t_1$ is such a cycle in G with $\ell > 2$ and suppose that t_1 and t_ℓ are the only consecutive elements conflicting in two variables, namely in v_1 and in some $u \in \{v_{\ell+1}, \dots, v_n\}$ [6]. Assume without loss of generality that $t_1, \dots, t_\ell$ behave as in Figure 9.3 and that $u \in t_1$ and $\overline{u} \in t_\ell$. (Note that neither t nor $t_2, \dots, t_{\ell-1}$ contains u or $\overline{u}$: if t contained u (resp. $\overline{u}$) it would conflict with t_ℓ (resp. t_1) in two variables; if any of $t_2, \dots, t_{\ell-2}$ (resp. $t_3, \dots, t_{\ell-1}$) contained u, it would conflict with t_ℓ (resp. t_1) in three variables; finally if t_2 (resp. $t_{\ell-1}$) contained $\overline{u}$ (resp. u), then it would conflict with t_1 (resp. t_ℓ) in two variables, contradicting the assumption of the lemma.) Then there is some partial assignment to the variables $\{v_{\ell+1}, \dots, v_n\} \setminus \{u\}$ consistent with $t_1, \dots, t_\ell$ and t. Denote one such by σ.

[5] Here it is used that $k < n$ and is assumed implicitely that every variable occurs in some of the terms of φ.

[6] If $\ell = 2$, then t_1 and t_ℓ does not conflict in v_1—which is the reason for handling this case separately.

Figure 9.4: The cycle $t_1, \ldots, t_\ell, t_1$. In the row of a term: "$+$" means that the given variable appears unnegated in it, "$-$" means that it appears negated in it, and " $\cdot$ " means that it does not appear in it. In the row of an assignment: "$+$" means that it assigns 1 to the given variable, "$-$" means that it assigns 0. Terms $t, t_1, \ldots, t_\ell$ do not conflict in other variables.

	v_1	v_2	v_3	$\cdots$	$v_{\ell-2}$	$v_{\ell-1}$	v_ℓ	u
t	$+$	$+$	$+$	$\cdots$	$+$	$+$	$+$	$\cdot$
t_1	$-$	$+$	$+$	$\cdots$	$+$	$+$	$\cdot$	$+$
t_2	$\cdot$	$-$	$+$	$\cdots$	$+$	$+$	$+$	$\cdot$
$\vdots$				$\ddots$				
t_ℓ	$+$	$+$	$+$	$\cdots$	$+$	$\cdot$	$-$	$-$
$\mathbf{x}$	$-$	$+$	$+$	$\cdots$	$+$	$+$	$+$	$-$
$\mathbf{y}$	$+$	$+$	$+$	$\cdots$	$+$	$+$	$-$	$+$

Let $\mathbf{x} := \sigma^{(v_2 \mapsto 1, \cdots, v_\ell \mapsto 1; v_1 \mapsto 0, u \mapsto 0)}$ (see Figure 9.4). Then one can make the following observations:

- $\overline{v_1} \in t_\mathbf{x}$, since $\mathbf{x} \notin \mathcal{T}(t)$ and $\mathbf{x}^{[v_1]} \in \mathcal{T}(t)$,
- $\overline{u} \in t_\mathbf{x}$, since $\mathbf{x} \notin \mathcal{T}(t_1)$ and $\mathbf{x}^{[u]} \in \mathcal{T}(t_1)$
- $v_\ell \notin t_\mathbf{x}$, as otherwise—definining $\mathbf{y} := \sigma^{(v_1 \mapsto 1, \cdots, v_{\ell-1} \mapsto 1; u \mapsto 1, v_\ell \mapsto 0)}$— $t_\mathbf{x}$ and $t_\mathbf{y}$ conflicts in three variables, because
  - $\overline{v_\ell} \in t_\mathbf{y}$, as $\mathbf{y} \notin \mathcal{T}(t)$ and $\mathbf{y}^{[v_\ell]} \in \mathcal{T}(t)$,
  - $v_1 \in t_\mathbf{y}$, as $\mathbf{y} \notin \mathcal{T}(t_1)$ and $\mathbf{y}^{[v_1]} \in \mathcal{T}(t_1)$,
  - $u \in t_\mathbf{y}$, as $\mathbf{y} \notin \mathcal{T}(t_\ell)$ and $\mathbf{y}^{[u]} \in \mathcal{T}(t_\ell)$.

  Consequently (as $t_\mathbf{x}$ conflicts with t in exactly one variable and does not contain v_ℓ) $t_\mathbf{x} \in V$ and $(t_\ell, t_\mathbf{x}), (t_\mathbf{x}, t_2) \in E$.
- $v_i \in t_\mathbf{x}$ for $i = 2, \ldots \ell - 1$, as otherwise $(t_i, t_\mathbf{x}) \in E$, which would mean that $t_2, \ldots, t_i, t_\mathbf{x}, t_2$ is a cycle in G shorter then ℓ—a contradiction.

But then $t_\mathbf{x}, t_2, \ldots, t_\ell, t_\mathbf{x}$ is a cycle of length ℓ (thus also of minimal length) such that all consecutive elements conflict in exactly one variable, contradicting Lemma 9.6. □

Based on the two previous Lemmas we can prove the following:

Lemma 9.8 *Assume* (9.3)*, let* $t = v_1 \cdots v_k$ *be a term of* φ *with* $k < n$*, and let* $G = G_{\varphi,t}$ *defined as in* (9.4)*. Then the smallest cycle in* G *has length at most two.*

Proof

Assume for the contradiction that $t_1, \ldots, t_\ell, t_1$ is a cycle in G of minimal length with $\ell > 2$. Assume furthermore *w.l.o.g.* that $t_1, \ldots, t_\ell, t_1$ is as in Figure 9.3. Then by the above lemmas there is some $1 \le i \le \ell - 1$ such that t_i and t_{i+1} conflict in two variables: in v_{i+1} and in some $u \in \{v_{k+1}, \ldots, v_n\}$. ($t$ contains neither u nor $\overline{u}$ as otherwise it would conflict with t_{i+1} or t_i in two variables.) Suppose i is the smallest such index. Then there is some partial assignment of the variables $\{v_1, \ldots, v_n\} \setminus \{v_i, v_{i+1}, u\}$ consistent with t, t_i and t_{i+1}. Denote one such by σ, and assume without loss of generality that t_i contains u, and t_{i+1} contains $\overline{u}$. (See Figure 9.5.)

Figure 9.5: Terms t_i, t_{i+1}, t and assignments $\mathbf{x}$ and $\mathbf{y}$.

	v_i	v_{i+1}	u
t	$+$	$+$	$\cdot$
t_i	$-$	$+$	$+$
t_{i+1}	$\cdot$	$-$	$-$
$\mathbf{x}$	$-$	$+$	$-$
$\mathbf{y}$	$+$	$-$	$+$

Let $\mathbf{x} := \sigma^{(v_i \mapsto 0, u \mapsto 0, v_{i+1} \mapsto 1)}$ and $\mathbf{y} := \sigma^{(v_i \mapsto 1, u \mapsto 1, v_{i+1} \mapsto 0)}$. Then

- $\overline{v_i} \in t_{\mathbf{x}}$, since $\mathbf{x} \notin \mathcal{T}(t)$ but $\mathbf{x}^{[v_i]} \in \mathcal{T}(t)$,
- $v_{i+1} \in t_{\mathbf{x}}$, since $\mathbf{x} \notin \mathcal{T}(t_{i+1})$ but $\mathbf{x}^{[v_{i+1}]} \in \mathcal{T}(t_{i+1})$,
- $\overline{u} \in t_{\mathbf{x}}$, since $\mathbf{x} \notin \mathcal{T}(t_i)$ but $\mathbf{x}^{[u]} \in \mathcal{T}(t_i)$,
- $\overline{v_{i+1}} \in t_{\mathbf{y}}$, since $\mathbf{y} \notin \mathcal{T}(t)$ but $\mathbf{y}^{[v_{i+1}]} \in \mathcal{T}(t)$, and
- $u \in t_{\mathbf{y}}$, since $\mathbf{y} \notin \mathcal{T}(t_{i+1})$ but $\mathbf{y}^{[u]} \in \mathcal{T}(t_{i+1})$.

Thus $t_{\mathbf{y}}$ does not contain v_i, as otherwise $t_{\mathbf{x}}$ and $t_{\mathbf{y}}$ would conflict in three variables. But then $t_{\mathbf{y}} \in V$, furthermore $(t_i, t_{\mathbf{y}}), (t_{\mathbf{y}}, t_{i+2}) \in E$, so $t_1, \ldots, t_i, t_{\mathbf{y}}, t_{i+2}, \ldots, t_\ell, t_1$ is also a cycle in G of minimal length, but with t_i and $t_{\mathbf{y}}$ conflicting only in one variable. That is, in this new cycle one gets further (starting from t_1) than in the original cycle without using an edge that's two endpoints conflict in two variables.

Iterating the above process if necessary, proceeding from the smaller indices to the larger ones, one obtains a cycle $t'_1, \ldots, t'_\ell, t'_1$ with consecutive elements conflicting in only one variable (apart maybe from t_ℓ and t_1), contradicting Lemma 9.7. □

Now all that is left to prove is that G contains no cycle of length 2.

Lemma 9.9 *Assume* (9.3), *let* $t = v_1 \cdots v_k$ *be a term of* φ *with* $k < n$, *and let* $G = G_{\varphi,t}$ *defined as in* (9.4). *Then* G *contains no cycle.*

Proof

By Lemma 9.8, as noted, it suffices to show that G contains no cycle of length 2. Assume for the contradiction that t_1, t_2, t_1 is a cycle in G and assume furthermore without loss of generality that $\overline{v_1} \in t_1$, $v_2 \notin t_1$, $v_1 \notin t_2$ and $\overline{v_2} \in t_2$. There are two cases: when t_1 and t_2 conflict in only one variable and when they conflict in two.

Let us consider the first case. Then t_1 and t_2 conflict in some $u \in \{v_{k+1}, \ldots, v_n\}$ (just like before, t cannot contain variable u, as otherwise it would conflict with t_1 or t_2

in at least two variables), and let us assume without loss of generality that $u \in t_1$ and $\overline{u} \in t_2$. Then there is some partial assignment to variables $\{v_3, \ldots, v_n\} \setminus \{u\}$ consistent with t_1 and t_2. Denote one such by σ. Let furthermore $\mathbf{x} := \sigma^{(v_1 \mapsto 0, u \mapsto 0, v_2 \mapsto 1)}$ and $\mathbf{y} := \sigma^{(v_1 \mapsto 1, u \mapsto 1, v_2 \mapsto 0)}$ (see Figure 9.6(a)). Using a similar argument as before one can see that $\overline{v_1}, v_2, \overline{u} \in t_{\mathbf{x}}$ and $v_1, \overline{v_2}, u \in t_{\mathbf{y}}$, thus the two terms conflict in three variables, contradiction.

Figure 9.6: Terms t_i, t_{i+1}, t and assignments $\mathbf{x}$ and $\mathbf{y}$.

	v_1	v_2	u
t	$+$	$+$	$\cdot$
t_1	$-$	$\cdot$	$+$
t_2	$\cdot$	$-$	$-$
$\mathbf{x}$	$-$	$+$	$-$
$\mathbf{y}$	$+$	$-$	$+$

(a)

	v_1	v_2	u	v
t	$+$	$+$	$\cdot$	$\cdot$
t_1	$-$	$\cdot$	$+$	$+$
t_2	$\cdot$	$-$	$-$	$-$
$\mathbf{x}$	$-$	$+$	$-$	$+$
$\mathbf{y}$	$+$	$-$	$+$	$-$

(b)

The second case is when t_1 and t_2 conflict in some $u, v \in \{v_{k+1}, \ldots, v_n\}$ (as in the previous case t contains neither u nor v). Let us assume without loss of generality that $u, v \in t_1$ and $\overline{u}, \overline{v} \in t_2$. Similarly as above, there is some partial assignment to variables $\{v_3, \ldots, v_n\} \setminus \{u, v\}$ consistent with t_1 and t_2; denote one such by σ, and put $\mathbf{x} := \sigma^{(v_1 \mapsto 0, u \mapsto 0, v \mapsto 1, v_2 \mapsto 1)}$ and $\mathbf{y} := \sigma^{(v_1 \mapsto 1, u \mapsto 1, v \mapsto 0, v_2 \mapsto 0)}$ (see Figure 9.6(b)). Again, one can show that $\overline{u}, \overline{v_1} \in t_{\mathbf{x}}$ and $u, \overline{v_2} \in t_{\mathbf{y}}$. Furthermore $v_2 \in t_{\mathbf{x}}$ (resp. $v_1 \in t_{\mathbf{y}}$), as otherwise $t_{\mathbf{x}} \in V$ (resp. $t_{\mathbf{y}} \in V$), and with t_2 (resp. with t_1) they would form a cycle of length two conflicting with each other in only one variable, which was ruled out in the previous case. Consequently $t_{\mathbf{x}}$ and $t_{\mathbf{y}}$ conflicts in three variables, contradiction. $\square$

The proof of the Theorem now follows from Claim 9.5 and Lemma 9.9, noting that if φ is a DDNF with conflict bound two that only has terms of length n, then $n \leq 2$, in which case the statement obviously holds.

9.4 Concluding Remarks

Theorem 9.1 considers a very limited class of DDNFs—for which a somewhat surprising property is proved. Nevertheless this does not bring us any closer to determining α_n^d in the general case, or to deriving a sharp bound for α_n. Finding answers to these problems requires further investigations.

Finally note that the results presented in this chapter—unless noted otherwise—appeared in the paper [119], authored by the author of the present dissertation.

Chapter 10

Decomposable Horn Formulas

Horn formulas (conjunctions of Horn clauses, i.e., clauses containing at most one unnegated literal—see Chapter 2) play a central role in artificial intelligence and in computer science. This formula class is attractive because it is expressive, allows for polynomial time inference, and indeed is generally computationally tractable. Accordingly it is one of the most studied Boolean formula classes.

In this chapter the following problem is considered:

Problem 10.1 *For Horn formulas φ and ψ, where ψ is a consequence of φ, when does there exist a proper Horn consequence χ of φ, such that $\psi \wedge \chi$ is equivalent to φ?*

Such a formula χ is called a φ-**complement** of ψ.

The motivation of this problem leads back to the topic of the first part of the present dissertation: to revision—or more precisely to belief revision.

Belief revision is interested in revising [1] a knowledge base in the presence of a new, potentially conflicting information, and usually approaches this problem by identifying postulates that should be satisfied by a rational revision operator, such as the AGM postulates [4], and characterizing operators that satisfy these postulates [45; 62]. In recent work, Flouris et al. [41] study belief revision in *general logics*, and formulate a property called **decomposability** of the logic. They show that decomposability is a necessary and sufficient condition for the existence of an AGM-compliant belief contraction operator. This framework is used in [42] to study decomposition properties of description logics, motivated by applications to the Semantic Web.

Problem 10.1 is, in fact, the reformulation of the above mentioned general decomposability problem for the class of Horn functions. Applying Horn functions to belief revision in [89] was intended to serve as a first step towards Horn-to-Horn belief revision: revision of Horn knowledge bases where the revised knowledge base is also required to be Horn. Horn-to-Horn belief revision is of interest for the efficient integration of

[1] Although the terminology is the same, in belief revision the notion of "revision" refers to a different kind of update method of the given system. However, as this serves only as a motivational background for the topic of the present chapter, it doesn't seem to be misleading to refer to this notion also as "revision". (On the other hand, when it is not clear from the context which notion is referred to as "revision", then it is made clear explicitely). Note furthermore that the original motivation for this work was exactly to bring theory revision and belief revision together.

various tasks facing a commonsense reasoning agent such as learning and revising its beliefs.

At this point it should mentioned that the class of Horn formulas has already been considered in theory revision (see [50; 52])—and of course also in learning (see [8; 44])—but, as noted in [89], the problem of belief revision that maintains a Horn knowledge base apparently has not been studied yet.

The main result of the chapter (Theorem 10.10) gives a complete answer to Problem 10.1 by giving two characterizations of all those pairs φ and ψ for which ψ has a φ-complement. The characterizations give efficiently decidable criteria and lead to efficient algorithms to construct a complement, if it exists. The complements constructed are only polynomially larger than the original knowledge base. As a corollary, one obtains a complete description of **decomposable** Horn formulas as well, where a Horn formula is decomposable if all its Horn consequences have a complement.

Problem 10.1 also has an interesting connection with another problem from a completely different field. Note that if ψ is a single Horn clause implicate C, then Problem 10.1 can be reformulated as follows: does φ have an irredundant conjunctive normal form expression containing C? According to Corollary 10.12 this problem is decidable in polynomial time. The related problem, studied by Hammer and Kogan [60], is that when C is a *prime* implicate and the irredundant conjunctive normal form expression is also assumed to consist of *prime* implicates only. In [60] such a prime implicate is called **non-redundant**, and is shown that non-redundancy is polynomially decidable for negative clauses, but is NP-complete for definite clauses.

Finally let us mention a related problem. Eiter and Gottlob [38] have shown that the problem, "Given Horn formulas φ, ψ and χ, is it the case that $\varphi' \wedge \psi \leq \chi$ for every maximal subformula φ' of φ consistent with ψ?" is co-NP-complete. This is a complexity-theoretic negative result for the revision method proposed by [39; 46], as formulas χ with the above property form the knowledge base obtained by revising the knowledge base φ with ψ.

10.1 Further Definitions and Notations

If a clause contains exactly one unnegated literal, then it is called **definite**, and if it contains none, it is called **negative**. A Horn formula is **definite Horn formula** if it consists of definite Horn clauses. A Boolean function is a **(definite) Horn function** if it has a (definite) Horn formula. It follows directly from the definitions that a Horn function f is definite if and only if $f(\mathbf{1}) = 1$.

For a Horn clause C, let its **body**, denoted $\mathrm{Body}(C)$, be the set of variables corresponding to the negative literals in C, or their conjunction (which will be clear from context). Also, let its **head**, denoted $\mathrm{Head}(C)$ be the unnegated variable of C if C is a definite clause, and 0 if C is a negative clause. The arrow symbol "$\rightarrow$" is used to denote the Boolean implication operator, so Horn clause C can be written as $\mathrm{Body}(C) \rightarrow \mathrm{Head}(C)$. For example, if C is the Horn clause $\overline{v} \vee \overline{w} \vee u$, then $\mathrm{Body}(C) =$

$\{v, w\}$, $\mathrm{Head}(C)$ is u, and C can also be written as $v, w \to u$ or $(v \wedge w) \to u$. If C is the Horn clause $\overline{v} \vee \overline{w}$ then it can also be written as $v, w \to 0$ or simply $v, w \to$.

Every clause that is an implicate of a definite Horn function is definite. Implication between Horn formulas can be decided in polynomial time (see, e.g., [79]).

A function f is **anti-monotone** if $\mathcal{T}(f)$ is downward closed, i.e., $f(\mathbf{x}) = 1$ and $\mathbf{y} \leq \mathbf{x}$ imply $f(\mathbf{y}) = 1$. This is equivalent to having a conjunctive normal form for it which consists of negative clauses. Horn functions have the following semantic characterization.

Theorem 10.2 ([71; 96]) *A Boolean function is Horn iff $\mathcal{T}(f)$ is closed under intersection.*

We will use a slight generalization of anti-monotone functions.

Definition 10.3 (almost anti-monotone function) *A function is almost anti-monotone if it is either anti-monotone, or there is an anti-monotone function g such that $\mathcal{T}(f) = \mathcal{T}(g) \cup \{\mathbf{1}\}$.*

The following is a direct consequence of Theorem 10.2.

Proposition 10.4 *Every almost anti-monotone function is Horn.*

Now we formulate the central concept discussed in this paper.

Definition 10.5 (f-complement) *For Horn functions f and g such that $f \leq g$, a Horn function h is an f-**complement of** g iff $f \lneq h$ and $f = (g \wedge h)$.*

Complements could also be defined assuming $f \lneq g$, but it is somewhat more convenient to formulate the definition as above. According to the definition, no f-complements exist if $f = 1$ (recall that 1 denotes the identically 1 function). This case is excluded from further consideration and we will always assume $f \neq 1$. Also according to the definition, $g = 1$ can never have a complement, so this case is also excluded from consideration in the following definition.

Definition 10.6 (decomposable Horn function) *A Horn function f is* **decomposable** *if every Horn consequence $g \neq 1$ of f has an f-complement.*

One usually works with formulas as opposed to functions, but as the notions of complement and decomposability depend only on the function represented by the formula, the definitions are given in a syntax-independent way.

10.2 Characterization of Decomposable Horn Formulas

Throughout the chapter let $\mathcal{V}' \subseteq \mathcal{V}$ denote the set of variables in focus.

For a function f and a set of variables $V \subseteq \mathcal{V}'$, we define the **f-closure of V** to be the set of variables

$$\mathrm{Cl}_f(V) = \{v \in \mathcal{V}' :\ f \leq (V \to v)\} \ .$$

Let us note a direct consequence of this definition.

Proposition 10.7 *If a negative clause C is an implicate of f, then $\mathrm{Cl}_f(\mathrm{Body}(C)) = \mathcal{V}'$.*

In order to formulate our main result, we need two definitions. The formula $\hat{\varphi}$ is obtained from φ by adding to the body of each definite clause in φ a variable not contained in the closure of its body, in all possible ways. For a Horn clause C of the form $\mathrm{Body}(C) \to \mathrm{Head}(C)$, we write $\mathrm{Body}(C), v \to \mathrm{Head}(C)$ for the Horn clause obtained from C by adding v to its body.

Definition 10.8 (body-building formula $\hat{\varphi}$) *For a Horn formula φ let $\hat{\varphi}$ be the formula*

$$\bigwedge_{C\in\varphi \text{ definite}} \ \bigwedge_{v\notin \mathrm{Cl}_\varphi(\mathrm{Body}(C))} (\mathrm{Body}(C), v \to \mathrm{Head}(C)).$$

Proposition 10.7 shows that we could have defined $\hat{\varphi}$ as a conjunction over all clauses of φ, as negative clauses make no contribution. Every clause of $\hat{\varphi}$ is definite. It may be the case that $\hat{\varphi}$ is the empty conjunction. This happens, for example, when φ consists of negative clauses only.

Given a Horn formula φ and a Horn clause D, we partition the clauses of φ not colliding with D into two classes.

Definition 10.9 (formulas $\mathcal{A}_\varphi(D)$ and $\mathcal{B}_\varphi(D)$) *Given a Horn formula φ and a Horn clause D, let*

$$\begin{aligned}\mathcal{A}_\varphi(D) &= \{C \in \varphi : C, D \text{ don't collide}, \mathrm{Body}(D) \subseteq \mathrm{Cl}_\varphi(\mathrm{Body}(C))\} \ ,\\ \mathcal{B}_\varphi(D) &= \{C \in \varphi : C, D \text{ don't collide}, \mathrm{Body}(D) \not\subseteq \mathrm{Cl}_\varphi(\mathrm{Body}(C))\} \ .\end{aligned}$$

The existence of a complement can now be characterized as follows.

Theorem 10.10 *Let $\varphi \not\equiv 1$ be a Horn formula, and ψ be a Horn consequence of φ. Then the following are equivalent:*

(a) *ψ has a φ-complement,*

(b) *$\hat{\varphi} \not\leq \psi$,*

(c) *for some clause D of ψ it holds that $\mathcal{B}_\varphi(D) \not\leq D$.*

Although the definition of $\hat{\varphi}$ is given in terms of a formula, it follows from this characterization that it actually depends on the function only (see also Lemma 10.20 below).

Corollary 10.11 (syntax-independence of $\hat{\varphi}$) *If φ_1 and φ_2 are equivalent Horn formulas then $\hat{\varphi}_1 \equiv \hat{\varphi}_2$.*

Theorem 10.10 is proved in the next section. Another proof of the first characterization (i.e., the equivalence of (a) and (b) in Theorem 10.10) is given in Section 10.4. The following corollary gives the algorithmic aspects of Theorem 10.10. It follows directly from the statement, resp., the proof(s) of the characterizations.

Corollary 10.12 *There is a polynomial time algorithm which, given a Horn formula φ and a Horn consequence ψ of φ, decides if ψ has a φ-complement, and if it does, then constructs such a φ-complement.*

The results are illustrated by the following simple example.

Example 10.1
Let $\mathcal{V}' = \{v, w, u\}$, $\varphi = C_1 \wedge C_2$, where $C_1 = (v \to w)$ and $C_2 = (w \to u)$. Then $\mathrm{Cl}_\varphi(\mathrm{Body}(C_1)) = \mathcal{V}'$ and $\mathrm{Cl}_\varphi(\mathrm{Body}(C_2)) = \{w, u\}$. So $\hat{\varphi} = (v, w \to u)$.

The clause $(v, w \to u)$ is implied by $\hat{\varphi}$, and so it has no φ-complement. This is also shown by the fact that $\mathcal{B}_\varphi(v, w \to u) = \{w \to u\}$, which implies $(v, w \to u)$.

On the other hand, the clause $(v \to u)$ is not implied by $\hat{\varphi}$, so it does have a φ-complement. This is also shown by the fact that $\mathcal{B}_\varphi(v \to u) = \{w \to u\}$, which does not imply $(v \to u)$. Both constructions described in the paper give the φ-complement $(v, u \to w) \wedge (w \to u)$.

Decomposable Horn functions have the following characterization.

Theorem 10.13 *For every Boolean function f the following are equivalent:*

(a) *f is a decomposable Horn function,*

(b) *there is a Horn representation φ of f such that $\hat{\varphi} \equiv 1$,*

(c) *for every Horn representation φ of f it holds that $\hat{\varphi} \equiv 1$,*

(d) *for every Horn implicate C of f it holds that $\mathrm{Cl}_f(\mathrm{Body}(C)) = \mathcal{V}'$,*

(e) *f is almost anti-monotone.*

Proof
The equivalence of (a), (b) and (c) follows directly from Theorem 10.10 and Corollary 10.11. The equivalence of (c) and (e) follows directly from the definitions.

(d) **implies** (e):
Assume that f is not almost anti-monotone, and let $\mathbf{x}, \mathbf{y}$ be truth assignments such that $\mathbf{y} \lneq \mathbf{x} \lneq \mathbf{1}$, $f(\mathbf{y}) = 0$ and $f(\mathbf{x}) = 1$. Then there is a Horn implicate C of f such that $C(\mathbf{y}) = 0$. As $C(\mathbf{x}) = 1$, it must be the case that C is a definite clause, $\mathrm{Body}(C)(\mathbf{y}) = \mathrm{Body}(C)(\mathbf{x}) = 1$, $\mathrm{Head}(C)(\mathbf{y}) = 0$ and $\mathrm{Head}(C)(\mathbf{x}) = 1$. As $\mathbf{x} \lneq \mathbf{1}$, there is a variable v such that $\mathbf{x}(v) = 0$. But then it must be the case that $v \notin \mathrm{Cl}_f(\mathrm{Body}(C))$, a contradiction.

(e) **implies** (d):
Assume that C is a Horn implicate of f and v is a variable such that $v \notin \mathrm{Cl}_f(\mathrm{Body}(C))$. Then C is a definite clause by Proposition 10.7. Let $\mathbf{x}$ be a truth assignment such that $f(\mathbf{x}) = 1$, $\mathrm{Body}(C)(\mathbf{x}) = 1$ and $\mathbf{x}(v) = 0$. As $f(\mathbf{x}) = 1$ it must be the case that $\mathrm{Head}(C)(\mathbf{x}) = 1$. Consider the truth assignment $\mathbf{y}$ obtained from $\mathbf{x}$ by switching the variable $\mathrm{Head}(C)$ off. Then $f(\mathbf{y}) = 0$. As $\mathbf{x}(v) = 0$, it holds that $\mathbf{x} \lneq \mathbf{1}$, so it follows that f is not almost anti-monotone. □

10.3 Proof of Theorem 10.10

We take care of the case where φ has negative implicates first.

Lemma 10.14 *Let $\varphi, \psi \not\equiv 1$ be Horn formulas such that $\varphi \leq \psi$, and ψ has a negative implicate D. Then*

- *ψ has a φ-complement,*
- *$\hat{\varphi} \not\leq \psi$,*
- *$\mathcal{B}_\varphi(D) \not\leq D$.*

Proof
It holds that $D(\mathbf{1}) = 0$, as D is negative. So $\varphi \leq \psi \leq D$ implies $\varphi(\mathbf{1}) = \psi(\mathbf{1}) = 0$. Let h be the Horn function that agrees with φ except that $h(\mathbf{1}) = 1$. We claim that h is a φ-complement of ψ. Clearly $\varphi \lneq h$ and so $\varphi \leq h \wedge \psi$. Now if $h(\mathbf{x}) = 1$, then either $\varphi(\mathbf{x}) = 1$ or $\mathbf{x} = \mathbf{1}$. Since $\psi(\mathbf{1}) = 0$, it follows that $h \wedge \psi \leq \varphi$, and hence $h \wedge \psi \equiv \varphi$ as desired.

Also, $\hat{\varphi}(\mathbf{1}) = 1$, because every clause of $\hat{\varphi}$ is definite (this includes the case when $\hat{\varphi}$ is empty), and therefore $\hat{\varphi} \not\leq \psi$. Similarly, Proposition 10.7 implies that every clause of $\mathcal{B}_\varphi(D)$ is definite, so $\mathcal{B}_\varphi(D)(\mathbf{1}) = 1$ and $\mathcal{B}_\varphi(D) \not\leq D$. □

We also need to construct a φ-complement of ψ. This is a special case of the construction of Section 10.4.

For the rest of the proof we may assume that ψ is a definite Horn formula. In order we will show: (a) implies (b), (b) implies (c), and (c) implies (a).

(a) **implies** (b):
This part is contained in Lemma 10.15, which, in turn, is split up into three lemmas. As these three lemmas do not actually refer to $\hat{\varphi}$, they are formulated in terms of functions rather then formulas.

Lemma 10.15 *Let $\varphi, \psi \not\equiv 1$ be Horn formulas such that ψ is definite and $\hat{\varphi} \leq \psi$. Then ψ does not have a φ-complement.*

Proof
The first of the three lemmas, Lemma 10.16, shows that clauses of $\hat{\varphi}$ have no φ-complement, and the second (resp., third) lemma extends this statement to $\hat{\varphi}$ (resp., consequences of $\hat{\varphi}$).

Lemma 10.16 *Let f be a Horn function and let $D_1 = (B \to z)$ and $D_2 = (B \to u)$ be definite Horn clauses with the same body B such that $f \leq D_1$ and $f \not\leq D_2$. Then*

$$D = (B, u \to z)$$

has no f-complement.

Proof
Assume that h is an f-complement of D. Thus, $f \leq h$, $h \not\leq f$ and $h \wedge D \leq f$. It then follows that

$$\begin{aligned} h &\not\leq D_1, & (10.1)\\ h &\not\leq D_2. & (10.2) \end{aligned}$$

Here (10.1) follows as otherwise $h \leq D_1 \leq D$ and so $h \leq h \wedge D \leq f$, and (10.2) follows as otherwise $f \leq h \leq D_2$.

Let $\mathbf{x}$ be the truth assignment which assigns 1 to the variables in $\mathrm{Cl}_h(B)$, and assigns 0 to all the other variables. Then $B(\mathbf{x}) = 1$ and we get from (10.1) and (10.2) that $u, z \notin \mathrm{Cl}_h(B)$, and so $\mathbf{x}(u) = \mathbf{x}(z) = 0$. Thus $D_1(\mathbf{x}) = 0$, implying $f(\mathbf{x}) = 0$, and it also holds that $D(\mathbf{x}) = 1$.

It remains to be shown that $h(\mathbf{x}) = 1$, as then $(h \wedge D)(\mathbf{x}) = 1$ and $f(\mathbf{x}) = 0$, contradicting the definition of the complement. Assume $h(\mathbf{x}) = 0$ and let D' be an implicate of h falsified by $\mathbf{x}$.

Case 1: D' is negative and it is a subclause of $B \to 0$. Then $h \leq D' \leq D_2$, contradicting (10.2).

Case 2: D' is negative and it is not a subclause of $B \to 0$. Then it contains negated variables $\overline{v_j}$, such that $v_j \notin B$ with $\mathbf{x}(v_j) = 1$ and hence $v_j \in \mathrm{Cl}_h(B)$ by the construction of $\mathbf{x}$. These can be 'resolved away' [2] using the implicates $B \to v_j$ of h, and we again get $h \leq (B \to 0) \leq D_2$.

Case 3: D' is definite. Then $\mathbf{x}$ assigns 0 to its head v, and so $v \notin \mathrm{Cl}_h(B)$. Variables $w \in (\mathrm{Cl}_h(B) \setminus B)$ in the body of D' can be 'resolved away' using the implicates $B \to w$ of h. We then get $h \leq (B \to v)$, contradicting $v \notin \mathrm{Cl}_h(B)$. □

Lemma 10.17 *If g_1 and g_2 have no f-complement then $g_1 \wedge g_2$ has no f-complement.*

Proof
Assume that h is an f-complement of $g_1 \wedge g_2$, that is, $f \leq h$, $h \not\leq f$ and $(h \wedge (g_1 \wedge g_2)) \leq f$. If $(h \wedge g_1) \leq f$ then h is an f-complement of g_1, a contradiction. Otherwise $(h \wedge g_1) \not\leq f$, and then $h \wedge g_1$ is an f-complement of g_2, again a contradiction. □

Lemma 10.18 *If $g_1 \leq g_2$ and g_1 has no f-complement, then g_2 has no f-complement.*

Proof
Assume that h is an f-complement of g_2, that is, $f \leq h$, $h \not\leq f$ and $h \wedge g_2 \leq f$. Then $h \wedge g_1 \leq h \wedge g_2 \leq f$, and so h is also an f-complement of g_1. □

[2] In this case and the next one it is convenient to refer to resolution but one could also argue directly about truth assignments as in the rest of the proof.

This completes the proof of Lemma 10.15. □

(b) **implies** (c):
For this part of the proof of Theorem 10.10 we show that if $\hat{\varphi} \not\leq D$ then $\bigwedge_{C \in \mathcal{B}_\varphi(D)} C \not\leq D$.

Let $\mathbf{x}$ be a truth assignment such that $\hat{\varphi}(\mathbf{x}) = 1$ and $D(\mathbf{x}) = 0$. Then it also holds that $\mathrm{Body}(D)(\mathbf{x}) = 1$ and $\mathrm{Head}(D)(\mathbf{x}) = 0$. It is sufficient to show that $C(\mathbf{x}) = 1$ for every $C \in \mathcal{B}_\varphi(D)$. By definition, there is a variable $v \in \mathrm{Body}(D) \setminus \mathrm{Cl}_\varphi(\mathrm{Body}(C))$. Thus $\mathrm{Body}(C), v \to \mathrm{Head}(C)$ is a clause of $\hat{\varphi}$ and therefore it is satisfied by $\mathbf{x}$. But $\mathrm{Body}(D)(\mathbf{x}) = 1$ implies $\mathbf{x}(v) = 1$, and so indeed $C(\mathbf{x}) = 1$.

(c) **implies** (a):
Let D be a clause in ψ such that $\bigwedge_{C \in \mathcal{B}_\varphi(D)} C \not\leq D$. We claim that $\mathcal{A}_\varphi(D) \neq \emptyset$. Consider an assignment $\mathbf{x}$ that satisfies $\bigwedge_{C \in \mathcal{B}_\varphi(D)} C$ but has $D(\mathbf{x}) = 0$. Now $\varphi \leq D$, so $\varphi(\mathbf{x}) = 0$. Thus there is some clause C of φ such that $C(\mathbf{x}) = 0$. As $D(\mathbf{x}) = 0$, the clauses C and D cannot collide; thus $C \in \mathcal{A}_\varphi(D)$.

Now we can define a φ-complement of ψ. For each clause $C \in \mathcal{A}_\varphi(D)$ let

$$\begin{aligned} \chi'_C &= \bigwedge_{z \in \mathrm{Body}(D)} (\mathrm{Body}(C) \to z), \\ \chi''_C &= (\mathrm{Body}(C), \mathrm{Head}(D) \to \mathrm{Head}(C)), \end{aligned}$$

and finally put

$$\chi = \left(\bigwedge_{C \in \mathcal{A}_\varphi(D)} \chi'_C \wedge \chi''_C \right) \wedge \left(\bigwedge_{C \in (\varphi \setminus \mathcal{A}_\varphi(D))} C \right).$$

Thus χ is formed from φ by replacing clauses $C \in \mathcal{A}_\varphi(D)$ by $\chi'_C \wedge \chi''_C$, and leaving the rest of the formula unchanged. Note that in the definition of χ''_C, if C is a negative clause then $\mathrm{Head}(C) = 0$. We claim that χ is a φ-complement of ψ.

$\varphi \leq \chi$: We need to show that for every $C \in \mathcal{A}_\varphi(D)$ it holds that $\varphi \leq \chi'_C$ and $\varphi \leq \chi''_C$. The definition of $\mathcal{A}_\varphi(D)$ implies $\mathrm{Body}(D) \subseteq \mathrm{Cl}_\varphi(\mathrm{Body}(C))$, thus for every $z \in \mathrm{Body}(D)$ it holds that $z \in \mathrm{Cl}_\varphi(\mathrm{Body}(C))$, and so every clause of χ'_C is an implicate of φ. It is obvious that $\varphi \leq \chi''_C$ as χ''_C is obtained from an implicate of φ by adding a literal to its body.

$\chi \not\leq \varphi$: It is sufficient to show that $\chi(\mathbf{x}) = 1$ for the truth assignment $\mathbf{x}$ above. As $D(\mathbf{x}) = 0$ and each clause in χ'_C and χ''_C collides with D, $\mathbf{x}$ satisfies χ'_C and χ''_C. The remaining clauses in χ come from φ: they either belong to $\mathcal{B}_\varphi(D)$ (in which case $\mathbf{x}$ satisfies them by definition), or they collide with D (and then $\mathbf{x}$ satisfies them as $D(\mathbf{x}) = 0$).

$\chi \wedge \psi \leq \varphi$: it is sufficient to show that for every $C \in \mathcal{A}_\varphi(D)$ it holds that $\chi'_C \wedge \chi''_C \wedge D \leq C$. Let $\mathbf{y}$ be any truth assignment satisfying $\chi'_C \wedge \chi''_C \wedge D$.

Let us assume first that C is definite. We need to show that if $\mathrm{Body}(C)(\mathbf{y}) = 1$ (which includes the case when $\mathrm{Body}(C)$ is empty), then $\mathrm{Head}(C)(\mathbf{y}) = 1$. But $\mathrm{Body}(C)(\mathbf{y}) = 1$ implies $\mathrm{Body}(D)(\mathbf{y}) = 1$ (which includes the case when $\mathrm{Body}(D)$ is empty). Hence $\mathrm{Head}(D)(\mathbf{y}) = 1$, and so (since $\chi''_C(\mathbf{y}) = 1$) it holds that $\mathrm{Head}(C)(\mathbf{y}) = 1$, as required. If C is negative then we need to show that $\mathrm{Body}(C)(\mathbf{y}) = 0$. Otherwise $\mathrm{Body}(D)(\mathbf{y}) = 1$, and so $\mathrm{Head}(D)(\mathbf{y}) = 1$ and thus $\chi''_C(\mathbf{y}) = 0$, a contradiction.

Example 10.2
Consider $\varphi = (v \to w) \wedge u$ and $\psi = u$. Then both clauses of φ are in $\mathcal{A}_\varphi(u)$, and so the φ-complement of ψ provided by the construction (after deleting redundant clauses) is $(v, w \to u)$.

10.4 Singleton Horn Extensions

We give a different proof of the equivalence (a) and (b) in Theorem 10.10, which also provides a semantic characterization of the body building formula. The proof is divided into two lemmas. Throughout the proof we use Theorem 10.2 without explicitly referring to it.

Lemma 10.19 *Let f, g be Horn functions such that $f \leq g$. Then g has an f-complement if and only if there is an $\mathbf{x} \in \mathcal{F}(f) \cap \mathcal{F}(g)$ such that $\mathcal{T}(f) \cup \{\mathbf{x}\}$ is a Horn function.*

Proof
The "if" direction follows by noting that $\mathcal{T}(f) \cup \{\mathbf{x}\}$ is an f-complement of g. For the "only if" direction assume that h is an f-complement of g. Let $\mathbf{x}$ be a minimal point (in the ordering defined by "$\leq$") in $\mathcal{T}(h) \setminus \mathcal{T}(f)$. Then since $h \wedge g \leq f$ it must be the case that $g(\mathbf{x}) = 0$. To show that $\mathcal{T}(f) \cup \{\mathbf{x}\}$ is a Horn function, assume that $\mathbf{x} \wedge \mathbf{y} \notin \mathcal{T}(f) \cup \{\mathbf{x}\}$ for some $\mathbf{y} \in \mathcal{T}(f)$. Then $\mathbf{x} \wedge \mathbf{y} \lneq \mathbf{x}$ and $h(\mathbf{x} \wedge \mathbf{y}) = 1$ would contradict the minimality of $\mathbf{x}$. □

The next lemma gives the semantic characterization of $\hat{\varphi}$. It shows that $\mathcal{T}(\hat{\varphi}) \setminus \mathcal{T}(\varphi)$ consists of precisely the **singleton Horn extensions** of φ, i.e., of those points which can be added to the set $\mathcal{T}(\varphi)$ maintaining the Horn property. This is a natural generalization of the minimal false points of an anti-monotone function.

Lemma 10.20 *Let φ be a Horn formula and $\mathbf{x} \in \mathcal{F}(\varphi)$. Then $\mathcal{T}(\varphi) \cup \{\mathbf{x}\}$ is a Horn function if and only if $\hat{\varphi}(\mathbf{x}) = 1$.*

Proof
First we prove the "only if" direction. Assume for contradiction that $\hat{\varphi}(\mathbf{x}) = 0$. Then there is a definite Horn implicate C of φ such that

$$(\mathrm{Body}(C), v \to \mathrm{Head}(C))(\mathbf{x}) = 0,$$

where

$$\varphi \not\leq (\mathrm{Body}(C) \to v). \tag{10.3}$$

Thus

$$\mathrm{Body}(C)(\mathbf{x}) = 1, \;\; \mathbf{x}(v) = 1 \text{ and } \mathrm{Head}(C)(\mathbf{x}) = 0. \tag{10.4}$$

According to (10.3), there is a truth assignment $\mathbf{y} \in \mathcal{T}(\varphi)$ falsifying $\mathrm{Body}(C) \to v$. Hence, taking into account that $\mathbf{y}$ must satisfy C, one has

$$\mathrm{Body}(C)(\mathbf{y}) = 1,\ \mathbf{y}(v) = 0 \text{ and } \mathrm{Head}(C)(\mathbf{y}) = 1. \tag{10.5}$$

Consider now the truth assignment $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$. From (10.4) and (10.5) we get

$$\mathrm{Body}(C)(\mathbf{z}) = 1,\ \mathbf{z}(v) = 0 \text{ and } \mathrm{Head}(C)(\mathbf{z}) = 0.$$

As $\mathbf{z}$ falsifies C, it holds that $\mathbf{z} \in \mathcal{F}(\varphi)$. Looking at the v-bits of $\mathbf{z}$ and $\mathbf{x}$ one gets $\mathbf{z} \lneq \mathbf{x}$, implying that $\mathcal{T}(\varphi) \cup \{\mathbf{x}\}$ is not closed under intersection, a contradiction.

Let us now prove the "if" direction. Assume for contradiction that $\mathcal{T}(\varphi) \cup \{\mathbf{x}\}$ is not Horn. Then there is a point $\mathbf{y} \in \mathcal{T}(\varphi)$ such that for $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ it holds that $\mathbf{z} \lneq \mathbf{x}$ and $\varphi(\mathbf{z}) = 0$. As $\mathbf{z} \leq \mathbf{y}$ and $\varphi(\mathbf{z}) \neq \varphi(\mathbf{y})$, it must also be the case that $\mathbf{z} \lneq \mathbf{y}$. Let C be a clause of φ falsified by $\mathbf{z}$. Then $C(\mathbf{y}) = 1$ and with $\mathbf{z} \lneq \mathbf{y}$ this implies that C is definite. As $\mathbf{z}$ falsifies C, it holds that

$$\mathrm{Body}(C)(\mathbf{z}) = 1 \text{ and } \mathrm{Head}(C)(\mathbf{z}) = 0.$$

Also, as $\mathbf{z} \lneq \mathbf{y}$, and $\mathbf{y}$ satisfies C

$$\mathrm{Body}(C)(\mathbf{y}) = 1 \text{ and } \mathrm{Head}(C)(\mathbf{y}) = 1.$$

As $\mathbf{z} \lneq \mathbf{x}$, and $\mathrm{Head}(C)(\mathbf{x}) = 1$ would imply $\mathrm{Head}(C)(\mathbf{z}) = \mathrm{Head}(C)(\mathbf{x} \wedge \mathbf{y}) = 1$, it follows that

$$\mathrm{Body}(C)(\mathbf{x}) = 1 \text{ and } \mathrm{Head}(C)(\mathbf{x}) = 0.$$

As $\mathbf{x}$ and $\mathbf{y}$ are incomparable, there is a variable u such that $\mathbf{x}(u) = 1$ and $\mathbf{y}(u) = 0$. Hence $\mathrm{Body}(C) \to u$ is falsified by $\mathbf{y}$, and so it is not an implicate of φ. Thus $\mathrm{Body}(C), u \to \mathrm{Head}(C)$ is a clause of $\hat{\varphi}$ falsified by $\mathbf{x}$, a contradiction. □

The "if" direction of Lemma 10.20 can also be proved by constructing a Horn formula for $\mathcal{T}(\varphi) \cup \{\mathbf{x}\}$ for every truth assignment $\mathbf{x} \in \mathcal{T}(\hat{\varphi}) \setminus \mathcal{T}(\varphi)$. Let C be a Horn clause falsified by $\mathbf{x}$ and $v \neq \mathrm{Head}(C)$ be a variable. Then let

$$\chi_C^v := \begin{cases} \mathrm{Body}(C), v \to \mathrm{Head}(C) & \text{if } v \notin \mathrm{Cl}_\varphi(\mathrm{Body}(C)), \\ \mathrm{Body}(C) \to v & \text{if } v \in \mathrm{Cl}_\varphi(\mathrm{Body}(C)),\ \mathbf{x}(v) = 1, \\ \mathrm{Body}(C), v \to \mathrm{Head}(C) & \text{if } v \in \mathrm{Cl}_\varphi(\mathrm{Body}(C)),\ \mathbf{x}(v) = 0. \end{cases}$$

Put

$$\chi_{\mathbf{x}} := \left(\bigwedge_{C(\mathbf{x})=0} \bigwedge_{v \neq \mathrm{Head}(C)} \chi_C^v \right) \wedge \left(\bigwedge_{C(\mathbf{x})=1} C \right).$$

Thus $\chi_{\mathbf{x}}$ is formed from φ by replacing clauses C falsified by $\mathbf{x}$ with $\bigwedge_{v \neq \mathrm{Head}(C)} \chi_C^v$, and leaving the rest of the formula unchanged. We claim that $\chi_{\mathbf{x}}$ is a Horn formula for

$\mathcal{T}(\varphi) \cup \{\mathbf{x}\}$. It is clear from the definitions that $\varphi \leq \chi_{\mathbf{x}}$ and $\chi_{\mathbf{x}}(\mathbf{x}) = 1$.

It remains to be shown that $\mathcal{T}(\chi_{\mathbf{x}}) \setminus \mathcal{T}(\varphi) = \{\mathbf{x}\}$. If $\mathbf{y}$ is a truth assignment with $\chi_{\mathbf{x}}(\mathbf{y}) = 1$ and $\varphi(\mathbf{y}) = 0$ then $\mathbf{y}$ must falsify a clause C of φ also falsified by $\mathbf{x}$. Thus $\mathrm{Body}(C)(\mathbf{x}) = \mathrm{Body}(C)(\mathbf{y}) = 1$ and $\mathrm{Head}(C)(\mathbf{x}) = \mathrm{Head}(C)(\mathbf{y}) = 0$. Now $\chi_{\mathbf{x}}(\mathbf{x}) = \chi_{\mathbf{x}}(\mathbf{y}) = 1$ implies that $\mathbf{x}(v) = \mathbf{y}(v) = 0$ for every $v \notin \mathrm{Cl}_{\varphi}(\mathrm{Body}(C))$, by considering the first case in the definition of χ_C^v. Similarly, $\mathbf{x}(v) = \mathbf{y}(v)$ for variables $v \in \mathrm{Cl}_{\varphi}(\mathrm{Body}(C)) \setminus \mathrm{Head}(C)$ follows by considering the second and third cases in the definition of χ_C^v.

Thus, given a consequence ψ of φ such that $\hat{\varphi} \not\leq \psi$, a φ-complement $\chi_{\mathbf{x}}$ of ψ can be constructed by first finding a truth assignment $\mathbf{x}$ with $\hat{\varphi}(\mathbf{x}) = 1, \varphi(\mathbf{x}) = 0$ and $\psi(\mathbf{x}) = 0$. Such a truth assignment can be found using a polynomial time Horn satisfiability algorithm in the usual manner. The formula $\chi_{\mathbf{x}}$ is then a φ-complement of ψ.

Example 10.3
Let $\varphi = (v \to w) \wedge u$ and $\psi = u$, as in Example 10.2. Then $\hat{\varphi} = (v, u \to w) \wedge (v \to u) \wedge (w \to u)$. So $\mathbf{0}$ is a truth assignment satisfying $\hat{\varphi}$ and falsifying φ and ψ, and the φ-complement of ψ provided by the construction (after deleting redundant clauses) is $(v \to w) \wedge (w \to u)$, which differs from the φ-complement of Example 10.2.

Both constructions presented for the complement may increase the size of the formula by a linear factor, and it is not known whether this increase is necessary. (Similar questions for DNFs are studied in [101].)

10.5 Concluding Remarks

Regarding the original motivation of the work presented in this chapter, the result that the only decomposable Horn formulas are the almost antimonotone ones are less satisfactory. The paper [89] proposes some directions to resolve this dilemma somehow. Related to the result of this chapter, the above paper also contains some experimental results about what fraction of implicates of a random Horn formula have complements.

Finally note that the results presented in this chapter—unless noted otherwise—appeared in the paper [89], co-authored by the author of the present dissertation.

Appendix A

Summary

Theory revision, as part of learning theory is interested in reconstructing some unknown function acquiring information about it via some protocol, specified by the given learning model. However, as opposed to the general learning problem, it is assumed that the learner is not new to the given task, but it initially has a hypotheses (in form of some formula) that is assumed to be some rough approximation of the unknown function. The efficiency criteria is that the running time is polynomial in the size of the different parameters, and that the amount of extra information, aquired via the protocol is also polynomial in the amount of information needed to represent the unknown function given the initial formula. In the first part of the dissertation theoretical results are considered from the field of theory revision.

In the second part characterizational results are presented; all showing equivalence between some syntactical and some semantical properties of some classes of Boolean functions.

Chapters 1–3 are introductory.

In Chapter 4 read-once functions are considered (a function is read-once function if it is representable with a formula in which every variable occurs at most once), discussing the corresponding results appeared in the paper [52]. The importance of this formula class is rather theoretical, being a nontrivial subclass of Boolean formulas that is tractable from several different aspects, and has a nice semantic characterization [58; 74; 102]. This class is shown to be efficiently learnable in the query model using membership and equivalence querie [13], which motivated the research aimed to construct an efficient algorithm for it. The main result of this chapter is a revision algorithm for this class in the deletions-only case (Algorithm `ReviseReadOnce`), which is shown to be an efficient revision algorithm (Theorem 4.7). Additionaly it was shown that the algorithm is optimal in the sense that both type of query used by Algorithm `ReviseReadOnce` is necessary for the efficiency (Theorem 4.13 and Theorem 4.14), and that the query complexity of any revision algorithm for this class is more or less of the same order of magnitude as that of Algorithm `ReviseReadOnce`—or worse (Theorem 4.11).

In Chapter 5 the revisability of Boolean threshold functions are considered, discussing the results appeared in the paper [116]. (A Boolean function is said to be a

threshold function if it can be represented by a set of variables R and a threshold θ, such that it evalutes 1 on exactly those assignments which assign 1 to at least θ of the variables in R.) Threshold functions (although in a more general form) are famous for being the basic ingredient of neural networks and support vector machines—and has several other applications as well. Boolean threshold functions are also known to be efficiently learnable in the query learning model [64] (however the learning algorithm presented in [64] uses only membership functions). The main result is again an algorithm (Algorithm `ReviseThreshold`) which is an efficient revision algorithm for the class of Boolean threshold functions in the query model (see Theorem 5.5). Again, it is also examined whether the query complexity of the algorithm is (more or less) as good as the optimal, and the answer was found to be positive (Proposition 5.8). In view of that the learning algorithm of Hegedűs for this class uses only membership queries, the question whether both type of queries are necessary for the efficient revision seems even more appropriate. However, as it is shown by Theorem 5.6 and Theorem 5.7, the answer is again positive. Finally it is shown that the natural extension of Algorithm `Winnow` [92] does not give an efficient revision algorithm for the class of threshold formulas (Proposition 5.9). This is interesting in view of that this algorithm is famous for learning some formula classes highly efficiently using some (general) threshold function representation.

As a closure of the first part dealing with theory revision, in Chapter 6 the revisability projective DNFs is considered, discussing the corresponding results appeared in [115]. Projective DNF formulas form a subclass of the disjunctive normal form formulas, introduced recently Valiant [128]. (The motivation for considering *subclasses* of the DNFs has substantially grown after the recent result of Alekhnovich *et al.* proving that, unless RP = NP, the class of DNFs is not efficient learnable [5].) This class was found by Valiant to be suitable for a special form of learning, called projective learning, the general behind it being that learning, similarly to other biological processes, should be carried out on multiple levels in a distributed manner. The main result of this chapter is that a natural extension (Algorithm `RevWinn`) of Valiant's algorithm is an efficent revision algorithm for the class of k-projective DNFs in the mistake bonded model (Theorem 6.3). The algorithm (just like the one used by Valiant [128]) consists of two levels. On the lower level simple learning algorithms are run, each concentrating on just a small part (or restriction) of the function to be learned. On the upper level another simple algorithm is run, which, on one hand, learns how to (re)combine the output of the algorithms on the lower level, and, on the other hand, it filters the information forwarded to these algorithms such that each one receives only that part of the information which is supposed to be relevant for it. In the second part of the Chapter a learnability related parameter, the so called exclusion dimension of the class is examined. This parameter is known to be related to the query complexity of the best learning algorithms for a given class (see [11; 67]) which, combined with the result on the exclusion dimension derived in the chapter implies the lower bound $\binom{\lfloor n/4 \rfloor}{k} - 1$ for the query complexity of this class (Proposition 6.9).

In Chapter 7 a further, characterization result is presented for projective DNF formulas, discussing the corresponding result appeared in [115]. Projective DNFs are defined in a rather semantic way (which is more apparent from part (a) of Lemma 6.5 from the preceding chapter), however the main result of this chapter, Theorem 7.3 gives a simple syntactic description for a subclass of this class, called 1-projective DNFs.

In Chapter 8 the relation between the number of terms in a DNF and the number of prime implicants of it is considered, discussing the results appeared in [114]. (A term t is an implicant of some Boolean function f, if any assignment saisfying t also satisfies f, meanwhile t is said to be a prime implicant of f if, in addition, this does not hold for any term obtained from t by removing some literals from it.) Section 8.3 discusses previoulsy known results on the topic: that if some DNF consists of K terms, then it has at most $2^K - 1$ prime implicants [31; 90; 97], and it is also mentioned that this bound is known to be sharp [88; 90; 97]. The results get completed in the subsequent sections by giving a charactarization DNFs that have as many prime implicants as this bound allows (Theorem 8.1). This is shown by reducing the problem to the following problem: if in some DNF tautology each pair of terms conflict in exactly one variable (i.e., each pair is resolvable) then it posesses a tree-like structure (i.e., there is some variable v appearing in each term; there is some variable w appearing in each term that contains v negated, and there is some variable u in each term that contains v unnegated; and so on).

Chapter 9 considers a generalization of the intermediate result in the previous chapter (about that DNF tautologies with terms conflicting in exactly one variable pairwise possess a tree-like structure), discussing the results appeared in [119]. More precisely in Theorem 9.1 it is shown that if in some DNF tautology each pair of terms conflict in at least one but at most two variables, then it also posesses a tree-like structure (also mentioning how it relates to various generalizations motivated by semantic resp. syntactic considerations). However, further relaxing the bound given for the conflict of the terms to three, the above mentioned tree-like structure will not be automatic—as is demonstrated by an example. This problem is also a special case of a problem considered in [93], that, given a DNF tautology, the task is to construct a decision tree such that for each term of the DNF generated by it there is a term of the tautology that is a subterm of it. They have shown that even for some very simple DNFs this problem requires a decision tree with extremely big complexity; however the result presented in this chapter implies that for each DNF in the above mentioned restricted class there exists always some simple decision tree [1].

Finally, in Chapter 10 decomposable Horn formulas are considered (conjunctive normal form formulas in which every clause contains at most one unnegated variable), discussing the results from [89]. Horn formulas, being an expressive class which also allows for polynomial time inference, and indeed is generally computationally tractable, play a central role in artificial intelligence and in computer science. The notion of decomposability comes from belief revision [2], a field interested in revising knowledge

[1]Actually the result states something stronger: for this restricted class basically the DNFs themselves can be considered as decision trees in some sense.

[2]Belief revision is related to theory revision (at least in it topic);thus—as a closure—the two main topics of the dissertation meet again.

base in such a manner that satisfies some "reasonability" properties, that are typically formulated in the form of postulates. Decomposability was introduced for general logics in [41], where it was also shown to be equivalent to the existence of some revision operator satisfying the AGM postulates [4]—one of the most popular postulates used in belief revision. The main result of the chapter is Theorem 10.10, showing characterizations for the existence of a complement of a Horn function consequence of another Horn function, which in turn provides a complete description of decomposable Horn formulas. The characterizations lead to efficient algorithms for the construction of a complement whenever it exists (which is in contrast with a related, but somewhat more stringent complement notion of [60], the existence of which is occasionally NP-complete to decide). The result, as is purely combinatiorial, but was meant in [89] as a first step towards what is referred to as "Horn-to-Horn belief revision": revision of Horn knowledge bases where the revised knowledge base is also required to be Horn; integrating hopefully efficient revision (the central notion in theory revision) and common sense reasoning (as a main goal in belief revision).

Appendix B

Összefoglalás

Az elméletrevízió – a tanuláselmélet részeként – azt vizsgálja, hogyan rekonstruálható hatékonyan valamely ismeretlen függvény különböző (az adott tanulási modell által meghatározott) protokollokon keresztül információt szerezve a függvényről. A tanulás szokásos alapszintuációjától eltérően azonban itt feltesszük, hogy a tanuló már rendelkezik valamilyen előismerettel erről a függvényről, pontosabban, hogy van egy kiinduló hipotézise (valamilyen formula képében), mely a tanulandó függvényt bizonyos értelemben jól közelíti. A futásidőre vonatkozó hatékonysági kritérium az, hogy legyen polinomálisan korlátos a probléma különböző paramétereinek méretében, az információelméleti pedig az, hogy a protokollon keresztül szerzett információ mennyisége legyen polinomiálisan korlátos azon információ mennyiségében, amennyivel az ismeretlen függvény leírható a kezdeti hipotézis ismeretében. A disszertáció első felében elméleti eredményeket tárgyaltunk az elméletrevízió témaköréből.

A disszertáció második felében karakterizációs eredményeket vizsgáltunk, melyek mind Boole-függvények valamely szemantikus illetve szintaktikus tulajdonságai között mutattak ekvivalenciát.

Az első három fejezet bevezető jellegű.

A 4. fejezetben read-once függvényekkel foglalkoztunk (egy függvény read-once – azaz egyszer olvasó –, ha reprezentálható olyan formulával, melyben minden változó legfeljebb egyszer fordul elő); ezen vizsgálatok alapjául az [52] cikk idevágó eredményei szolgáltak. Ezen függvényosztály elméleti szempontból igen jelentős, tekintve, hogy Boole-függvényeknek egy olyan, nemtriviális részhalmaza, melynek elemei (sok tekintetben) algoritmikusan hatékonyan kezelhetők, ráádasul egy kellemes szemantikus karakterizációja is ismert [58; 74; 102]. A függvényosztályról az is ismert (lásd [13]), hogy hatékonyan tanulható az úgynevezett query model (tanulás kérdések által) keretein belül, ha a tanuló használhat mind membership query-t (értékre kérdezés) mind equivalence query-t (ekvivalenciára kérdezés). A fejezet fő eredménye, hogy az ott ismertetett `ReviseReadOnce` algoritmus a függvényosztály hatékony revízióját valósítja meg a csak-törléses esetben (lásd a 4.7. tételt). További eredményként ismertetésre került, hogy az algoritmusban használatos két kérdéstípus bármelyikét mellőzve a függvényosztály revíziója nem valósítható meg hatékonyan (lásd a 4.13 és a 4.14 tételeket), illetve hogy az algoritmus által használt kérdések mennyisége nagyságrendileg lényegében szintén optimális (lásd a 4.11 tételt).

Az 5. fejezetben küszöbfüggvényekkel foglalkoztunk; ezen vizsgálatok alapjául a [116] cikk idevágó eredményei szolgáltak. (Egy függvényt küszöbfüggvénynek tekintünk, ha reprezentálható változók egy R halmazával és egy θ küszöb értékkel olyan módon, hogy a függvény pontosan azon értékadások esetén ad 1-et, melyek az R-beli változók közül legalább θ-hoz 1-et rendelnek értékül.) A küszöbfüggvények jelentőségét jelzi, hogy (habár a fentinél általánosabb formában megadva) a mesterséges neuronhálók illetve SVM-ek (support vector machine-ek) egyik alap építőköveként használatosak. Küszöbfüggvényekről is ismert, hogy hatékonyan tanulhatók a query model keretein belül, ám ezen függvényosztály esetén ehhez elég csak a membership query-k használata (lásd a [64] cikkben ismertetett algoritmust). A fejezet fő eredménye, hogy az ott ismertetett `ReviseThreshold` algoritmus a függvényosztály hatékony revízióját valósítja meg az általános esetben (lásd az 5.5. tételt). Ezen felül megintcsak bizonyításra került, hogy az algoritmus által használt kérdések mennyisége nagyságrendileg lényegében optimális (lásd az 5.8. állítást). Figyelembe véve, hogy – amint az fent említetésre került – a függvényosztály hatékonyan tanulható csak membership query-k használatával is, ebben az esetben még aktuálisabb a kérdés, hogy vajon a hatékony revízióhoz szükség van-e mindkettőre. A válasz, mint azt az 5.6. és 5.7. tételek mutatják, igenlő. Végezetül megmutattuk, hogy Littlestone híres `Winnow` Algoritmusa (mely a [92] cikkben került ismertetésre), illetve annak egy megfelelő, természetes elméletrevíziós kiterjesztése nem hatékony revíziós algoritmus ezen függvényosztályra. Ez azért is meglepő, mert ezen algoritmus az által vált híressé, hogy bizonyos függvények tanulását kimagaslóan hatékonyan valósítja meg, és ráadásul (általánosabb értelemben vett) küszöbfüggvényként reprezentálja a mindenkori hipotézisét.

A 6. fejezetben, az elméletrevízióval foglalkozó első rész zárásaként projektív DNF formulákkal foglalkoztunk; ezen vizsgálatok alapjául a [115] cikk idevágó eredményei szolgáltak. A diszjunktív normálformájú formulák részosztályát alkotó projektív DNF formulák Valiant [128] cikkében kerültek bevezetésre. (A DNF-ek különböző részosztályainak vizsgálata azáltal kapott még nagyobb hangsúlyt, hogy Alekhnovich-ék [5] cikkükben megmutatták, hogy – hacsak az NP és RP osztályok nem egyenlőek – a DNF-ek osztálya nem tanulható hatékonyan.) A formulaosztály jelentőségét az szolgáltatta Valiant számára, hogy alkalmasnak bizonyultak az ún. projektív tanulásra, melynek lényege, hogy a tanulás, a biológiában megfigyelhető más folyamatokhoz hasonlóan, több szinten, osztott módon történik. A fejezet fő eredménye, hogy az ott ismertetett `RevWinn` algoritmus, mely Valiant tanulóalgoritmusának egy természetes kiterjesztése, a projektív DNF formulák hatékony revízióját valósítja meg a mistake bounded model (hibakorlátozott modell) keretein belül (lásd a 6.3. tételt). Az algoritmus, Valiant eredeti algoritmusához hasonlóan, két szintből tevődik össze. Az alsó szinten egy egyszerű tanulóalgoritmus több példánya kerül futtatásra, melyek mind a tanulandó függvény (pontosabban a függvény értelmezési tartományának) egy kis szeletére (avagy projekciójára) figyelnek csak. A felső szinten megintcsak egy egyszerű tanulóalgoritmus használatos (ezúttal viszont csak egy darab), egyrészt azzal a céllal, hogy megtanulja, hogyan kell az alsó szinten futtatott egyszerű tanulóalgoritmusok által reprezentált hipotéziseket összekapcsolni, másrészt azzal, hogy – megszűrve az információt – az alsó szinten lévő minden egyes algoritmusnak csak az ő számára releváns információt továbbítsa. A fejezet második részében a formulaosztály egy, a tanulással kapcsolatos paraméterét, az ún. exclusion dimension (kizárási dimenzió) paraméterét vizsgáltuk (ezen paraméterről bővebben lásd a [11; 67] cikkeket). Ezen eredményt, valamint az exclusion dimension és a query complexity (kérdési bonyolultság) között fennálló, ismert összefüggéseket felhasználva megmutatjuk, hogy a formulaosztály nem tanulható kevesebb mint $\binom{\lfloor n/4 \rfloor}{k} - 1$ kérdést használva (a legrosszabb eset analízisben).

A 7. fejezetben további, karakterizációs jellegű kérdéseket vizsgáltunk a projektív DNF formulákkal kapcsolatosan; ezen vizsgálatok alapjául a [115] cikk idevágó eredménye szolgált. A projektív DNF-ek szemantikus jellegű módon lettek definiálva (melyet talán a 6.5 Lemma (a) pontja hangsúlyoz ki a leglátványosabban), ezért is érdekes a 7.3 tétel eredménye, mely ezen formulaosztály egy részosztályának, az 1-projektív DNF-eknek egy szemantikus leírását adja meg.

A 8. fejezetben egy DNF termjeinek illetve prímimplikánsainak száma közti kapcsolatot vizsgáltuk; ezen vizsgálatok alapjául a [114] cikk eredményei szolgáltak. (Egy t term implikánsa egy Boole függvénynek, a t-t kielégítő értékadások a függvényt is mind kielégítik, illetve prímimplikánsa, ha ez a tulajdonság már egy olyan termre sem teljesül, melyet t-ből literálok elhagyásával kaphatunk.) A 8.3. részben a témában ismert korábbi eredményeket ismertettük (a teljesség kedvéért bizonyítással együtt); többek közt azt, hogy egy K tagú DNF-nek legfeljebb $2^K - 1$ prímimplikánsa lehet [31; 90; 97], és hogy ez a korlát éles [88; 90; 97]. A fejezet fő eredménye, hogy teljes karakterizációját adja azon DNF-eknek, melyek prímimplikánsainak száma eléri ezt a felső korlátot (lásd a 8.1. tételt). A bizonyítás során a problémát visszavezettük arra, hogy ha egy DNF tautológiában minden tag minden másik taggal pontosan egy változóban ütközik, akkor a DNF-nek egy speciális fa struktúrája van.

A 9. fejezetben azon probléma került általánosításra, melyre az előző fejezet eredeti problémája vissza lett vezetve; ezen vizsgálatok alapjául a [119] cikk eredményei szolgáltak. Pontosabban azt mutattuk meg (lásd a 9.1. tételt), hogy ha egy DNF-ben minden tag minden másik taggal legalább egy, de legfeljebb két változóban ütközik, akkor szintén rendelkezik a fent említett fa jellegű struktúrával, de ha a megengedett ütközések számát már háromra növeljük, akkor ez a struktúra már nem jelenik meg minden esetben. Kifejtére került továbba az is, hogy ez az eredmény hogyan viszonyul különböző további, szemantikus illetve szintaktikus megfontolások által vezérelt általánosításokhoz. Megemlítettük azt is, hogy ez a probléma egy speciális esete a [93] cikkben tárgyalt problémának, mely azzal foglalkozik, hogy egy adott DNF tautológia esetén mekkora a legkisebb olyan döntési fa, amely olyan DNF tautológiát generál, aminek minden tagja az adott DNF valamely termjének kibővítése plusz literálokkal.

Végezetül, a 10. fejezetben ún. felbontható Horn fromulákat vizsgáltunk (Horn formula egy olyan CNF, amelyben minden klóz legfeljebb egy negálatlan változót tartalmaz); ezen vizsgálatok alapjául a [89] cikk eredményei szolgáltak. A Horn formulák igen fontos szerepet játszanak a mesterséges intelligenciában, illetve általában a számítástudományban, melynek alapja, hogy a formulaosztály kifejezőképessége relatíve igen jó, és emellett algoritmikusan hatékonyan kezelhető. A felbonthatóság fogalma a belief revision témaköréből származik [1], mely témakör főként tudásbázisok (hétköznapi értelemben vett) racionalitási tulajdonságokat teljesítő revíziójával foglalkozik, melyeket tipikusan posztulátumok formájában fogalmaznak meg. A felbonthatóság fogalmát általános logikákra fogalmazták meg a [41] cikkben, ahol megmutatták, hogy az AGM posztulátumok [4] (a legismertebb posztulátumok egyike a témakörben) teljesülésének szükséges és elégséges feltétele, hogy az adott logikában létezzen felbontható revíziós operátor. A fejezet fő eredményeként a 10.10. tételben karakterizáltuk, hogy milyen esetkben van egy Horn formulának egy másik (őt implikáló) Horn formulára nézve komplemense. Ezt felhasználva végül megadtuk a felbontható Horn formulák egy jellemzését. Mint megmutattuk, ha létezik, a komplemens hatékony konstruálható, szemben az irodalolmban egy korábban vizsgált, valamelyest szigorúbb komplemens fogalommal, melynek meglétének eldöntése bizonyos esetekben NP-nehéz. Az eredmény a közölt formában pusztán kombinatorikai jellegű, ám mindez egy Horn formula alapú módszer első lépéseként került vizsgálatra a a [89] cikkben, melynek jövőbeni célja a hatékony revízió ötvözése a belief revison által vizsgált racionalitási tulajdonságokkal.

[1]Ezen témakör rokon az elméletrevízióval, így a disszertáció végén egy fejezet erejéig bizonyos értelemben újra találkozik a disszertáció két fő témája.

Bibliography

[1] R. Aharoni and N. Linial. Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. *Journal of Combinatorial Theory Series A*, 43(2):196–204, 1986.

[2] H. Aizenstein, T. Hegedűs, L. Hellerstein, and L. Pitt. Complexity theoretic hardness results for query learning. *Computational Complexity*, 7(1):19–53, 1998.

[3] H. Aizenstein, L. Hellerstein, and L. Pitt. Read-thrice DNF is hard to learn with membership and equivalence queries. In *Proc. 33rd Symposium on Foundations of Computer Science (FOCS 1992)*, pages 523–532. IEEE Computer Society Press, 1992.

[4] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.

[5] M. Alekhnovich, M. Braverman, V. Feldman, A. R. Klivans, and T. Pitassi. Learnability and automatizability. In *Proc. 45th Symposium on Foundations of Computer Science (FOCS 2004)*, pages 621–630. IEEE Computer Society Press, 2004.

[6] D. Angluin. Learning propositional Horn sentences with hints. Technical Report YALEU/DCS/RR-590, Department of Computer Science, Yale University, Dec. 1987.

[7] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[8] D. Angluin. Learning with hints. In *Proc. 1st. Workshop on Computational Learning Theory (COLT 1988)*, pages 167–181. Morgan Kaufmann, 1988.

[9] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, 1990.

[10] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[11] D. Angluin. Queries revisited. *Theoretical Compututer Science*, 313(2):175–194, 2004. Earlier version appeared in 12th ALT, 2001.

[12] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9(2–3):147–164, 1992.

[13] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *Journal of the ACM*, 40(1):185–210, 1993.

[14] D. Angluin and M. Kharitonov. When won't membership queries help? *Journal of Computer and Systems Sciences*, 50(2):336–355, 1995. Earlier version appeared in 23rd STOC, 1991.

[15] N. H. Arai, T. Pitassi, and A. Urquhart. The complexity of analytic tableaux. In *Proc. 33rd Symposium on Theory of Computing (STOC 2001)*, pages 356–363, ACM Press, 2001.

[16] S. Argamon-Engelson and M. Koppel. Tractability of theory patching. *Journal of Artificial Intelligence Research*, 8:39–65, 1998.

[17] P. Auer and P. M. Long. Structural results about on-line learning models with and without queries. *Machine Learning*, 36(3):147–181, 1999.

[18] P. Auer and M. K. Warmuth. Tracking the best disjunction. *Machine Learning*, 32(2):127–150, 1998. Earlier version appeared in 36th FOCS, 1995.

[19] L. Babai and P. Frankl. *Linear algebra methods in combinatorics*. Preliminary version 2, Available from Univ. of Chicago Computer Science Dept., 1992.

[20] R. Bennet. *Improved Learning with Corrupt Oracles*. Master's thesis, Technion University, 2005.

[21] L. Bisht, N. H. Bshouty, and L. Khoury. Learning with errors in answers to membership queries. In *Proc. 45th Symposium on Foundation of Computer Science (FOCS 2004)*, pages 611–620, 2004.

[22] A. Blum. On-line algorithms in machine learning. Available from `http://www-2.cs.cmu.edu/~avrim/Papers/pubs.html`, 1996.

[23] A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9:373–386, 1992.

[24] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *Journal of Computer and Systems Sciences*, 50(1):32–40, 1995. Earlier version in 4th COLT, 1991.

[25] A. Blum and S. Rudich. Fast learning of k-term DNF formulas with queries. *Journal of Computer and Systems Sciences*, 51(3):367–373, 1995.

[26] N. Bshouty. Exact learning Boolean function via the monotone theory. *Information and Computation*, 123:146–153, 1995.

[27] N. Bshouty and L. Hellerstein. Attribute-efficient learning in query and mistake-bound models. *Journal of Computer and Systems Sciences*, 56(3):310–319, 1998.

[28] N. H. Bshouty and R. Cleve. On the exact learning of formulas in parallel. In *Proc. 33rd Symposium on the Foundations of Computer Sciences (FOCS 1992)*, pages 513–522. IEEE Computer Society Press, 1992.

[29] N. H. Bshouty, S. A. Goldman, T. R. Hancock, and S. Matar. Asking questions to minimize errors. *Journal of Computer and Systems Sciences*, 52(2):268–286, 1996. Earlier version in 6th COLT, 1993.

[30] L. Carbonara and D. Sleeman. Effective and efficient knowledge base refinement. *Machine Learning*, 37(2):143–181, 1999.

[31] A. K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24:7–11, 1978.

[32] P. Damaschke. Adaptive versus nonadaptive attribute-efficient learning. *Machine Learning*, 41(2):197–215, 2000.

[33] G. Davydov, I. Davydova, and H. Kleine Büning. An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Annals of Mathematics and Artificial Intelligence*, 23:229–245, 1998.

[34] L. De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201, 1997.

[35] R. Dechter and J. Pearl. Structure identification in relational data. *Artificial Intelligence*, 58(1–3):237–270, 1992.

[36] A. Dhagat and L. Hellerstein. PAC learning with irrelevant attributes. In *Proc. 35rd Annual Symposium on Foundations of Computer Science (FOCS 1994)*, pages 64–74. IEEE Computer Society Press, 1994.

[37] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.

[38] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2–3):227-270, 1992.

[39] R. Fagin, J. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *Proc.2nd, Symposium on Principles of Database Systems (PODS 1983)*, pages 352–365, ACM Press, 1983.

[40] R. Feldman. *Probabilistic Revision of Logical Domain Theories*. PhD thesis, Cornell University, 1993.

[41] G. Flouris, D. Plexousakis, and G. Antoniou. On generalizing the AGM postulates: preliminary results and applications. In *Proc. 10th Workshop on Non-Monotonic Reasoning (NMR 2004)*, pages 171–179, 2004.

[42] G. Flouris, D. Plexousakis, and G. Antoniou. Updating description logics using the AGM theory. In *7th Symposium on Logical Formalizations of Commonsense Reasoning*, 2005.

[43] M. Frazier. *Matters Horn and Other Features in the Computational Learning Theory Landscape: The Notion of Membership*. PhD thesis, University of Illinois at Urbana-Champaign, 1994. Technical report UIUCDCS-R-94-1858.

[44] M. Frazier and L. Pitt. Learning from entailment: An application to propositional Horn sentences. In *Proc. 10th International Confenrence on Machine Learning (ICML 1993)*, pages 120–127. Morgan Kaufmann, June 1993.

[45] P. Gärdenfors. *Knowledge in Flux*. Bradford Books/MIT Press, 1988.

[46] M. L. Ginsberg. Counterfactuals. *Artificial Intelligence*, 35–79, 1986.

[47] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[48] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[49] J. Goldsmith and R. H. Sloan. More theory revision with queries. In *Proc. 32nd Symposium on Theory of Computing (STOC 2000)*, pages 441–448. ACM Press, 2000.

[50] J. Goldsmith and R. H. Sloan. New Horn Revision Algorithms. *Journal of Machine Learning Research*, 6:1919–1938, 2005.

[51] J. Goldsmith, R. H. Sloan, B. Szörényi, and Gy. Turán. Improved algorithms for theory revision with queries. In *Proc. 13th Conference on Computational Learning Theory (COLT 2000)*, pages 236–247. Morgan Kaufmann, 2000.

[52] J. Goldsmith, R. H. Sloan, B. Szörényi, and Gy. Turán. Theory revision with queries: Horn, read-once, and parity formulas. *Artificial Intelligence*, 156:139–176, 2004.

[53] J. Goldsmith, R. H. Sloan, and Gy. Turán. Theory revision with queries: DNF formulas. *Machine Learning*, 47(2–3):257–295, 2002.

[54] R. L. Graham and H. O. Pollak. On the addressing problem for loop switching. *Bell System Technical Journal*, 50(8):2459–2519, 1971.

[55] D. A. Gregory, V. L. Watts, and B. L. Shader. Biclique decompositions and hermitian rank. *Linear Algebra and its Applications*, 292:267–280, 1999.

[56] R. Greiner. The complexity of revising logic programs. *Journal of Logic Programming*, 40:273–298, 1999.

[57] R. Greiner. The complexity of theory revision. *Artificial Intelligence*, 107(2):175–217, 1999.

[58] V. Gurvich. On repetition-free Boolean functions. *Uspekhi Matematicheskikh Nauk*, 32(1):183–184, 1977. (In Russian).

[59] P. L. Hammer and A. Kogan. Quasi-acyclic propositional Horn knowledge bases: optimal compression. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):751–762, 1995.

[60] P. L. Hammer and A. Kogan. Essential and Redundant Rules in Horn Knowledge Bases. In *Proc. 28th Hawaii International Conference on System Sciences (HICSS 1995*, 209–218, 1995.

[61] P. L. Hammer, A. Kogan, and U. G. Rothblum. Evaluation, strength and relevance of variables of Boolean functions. *SIAM Journal on Discrete Mathematics*, 13:302–312, 2000.

[62] S. O. Hansson. *A Textbook on Belief Dynamics: Theory Change and Database Updating*. Kluwer, 1999.

[63] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of Models for Polynomial Learnability. *Information and Computation*, 95(2):129–161, 1991.

[64] T. Hegedűs. On training simple neural networks and small-weight neurons. In *Proc. 1st Eurpean Conference on Computational Learning Theory (EuroColt 1993)*, pages 69–82. Oxford University Press, 1994.

[65] T. Hegedűs. Generalized teaching dimensions and the query complexity of learning. In *Proc. 8th Conference on Computational Learning Theory*, pages 108–117. ACM Press, 1995.

[66] T. Hegedűs and P. Indyk. On learning disjunctions of zero-one threshold functions with queries. In *8th International Workshop on Algorithmic Learning Theory (ALT 1997)*, vol. 1316 of LNAI, pages 446–460. Springer, 1997.

[67] L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan, and D. Wilkins. How many queries are needed to learn? *Jourrnal of the ACM*, 43(5):840–862, 1996.

[68] L. Hellerstein and V. Raghavan. Exact learning of DNF formulas using DNF hypotheses. *Journal of Computer and Systems Sciences*, 70:435–470, 2005.

[69] D. Helmbold, R. Sloan, and M. K. Warmuth. Learning nested differences of intersection closed concept classes. *Machine Learning*, 5(2):165–196, 1990. Earlier version appeared in 2nd COLT, 1989.

[70] S. Hoory and S. Szeider. A note on unsatisfiable k-CNF formulas with few occurrences per variable. *SIAM Journal on Discrete Mathematics*, 20:523–528, 2006.

[71] A. Horn. On sentences which are true on direct unions of algebras. *Journal of Symbolic Logic*, 16:14–21, 1951.

[72] S. Jukna. *Extremal Combinatorics*. Springer, 2001.

[73] S. Jukna, A. Razborov, P. Savický, and I. Wegener. On P versus NP ∩ co-NP for decision trees and read-once branching programs. *Computational Complexity*, 8(4):357–370, 1999.

[74] M. Karchmer, N. Linial, I. Newman, M. Saks, and A. Wigderson. A combinatorial characterization of read-once formulae. *Discrete Mathematics*, 114:275–282, 1993.

[75] J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions. In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS 1988)*, pages 68–80, 1988.

[76] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of Boolean formulae. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 285–294. ACM Press, 1987.

[77] M. Kearns and L. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

[78] M. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[79] H. Kleine Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms*. Cambridge University Press, 1999.

[80] J. Kivinen and M. K. Warmuth. Additive versus exponentiated gradient updates for linear prediction. In *Proc. 27th Symposium on Theory of Computing (STOC 1995)*, pages 209–218. ACM Press, 1995.

[81] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, second edition, 1978.

[82] M. Koppel, R. Feldman, and A. M. Segre. Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research*, 1:159–208, 1994.

[83] J. Kratochvíl, P. Savický, and Zs. Tuza. One more occurrence of variables makes satisfiability jump from trivial to NP-complete. *SIAM Journal on Computing*, 22:203–210, 1993.

[84] O. Kullmann. An application of matroid theory to the SAT problem. In *15th Conference on Computational Complexity (COCO 2000)*, pages 116–124. IEEE Computer Society, 2000.

[85] O. Kullmann. The combinatorics of conflicts between clauses. In *Proc. 6th Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, vol.2919 of LNCS, pages 426–440. Springer, 2003.

[86] O. Kullmann. On the conflict matrix of clause-sets. *Technical Report CSR 7-2003*, University of Wales at Swansea, 2003.

[87] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum In *Proc. 23rd Symposium on Theory of Computing (STOC 1991)*, pages 455–464. ACM Press, 1991.

[88] J.-M. Laborde. Sur le cardinal maximum de la base complète d'une fonction booléenne, en fonction du nombre de conjunctions de l'une de ses formes normales. *Discrete Mathematics*, 32:209–212, 1980.

[89] M. Langlois, R. H. Sloan, B. Szörényi, and Gy. Turán. Horn formulas, decomposability and Belief Revision. Manuscript.

[90] A. A. Levin. Comparative complexity of disjunctive normal forms. *Metody Discret. Analiz.*, 36:23–38, 1981. (In Russian)

[91] N. Littlestone. A mistake-bound version of Rivest's decision-list algorithm. Personal communication to Avrim Blum, 1989.

[92] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[93] L. Lovász, M. Naor, I. Newman, and A. Wigderson. Search problems in the decision tree model. *SIAM Journal on Discrete Mathematics*, 8:119–132: 1995.

[94] W. Maass and Gy. Turán. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9(2–3):107–145, 1992.

[95] J. A. Makowsky. Model theory and computer science: An appetizer. In *Handbook of Logic in Computer Science, Volume 1 (Background: Mathematical Structures)*, pages 763–814. Oxford University Press, 1992.

[96] J. C. C. McKinsey. The decision problem for some classes without quantifiers. *Journal of Symbolic Logic*, 8:61–76, 1943.

[97] C. McMullen and J. Shearer. Prime implicants, minimum covers, and the complexity of logic simplification. *IEEE Transactions on Computers*, 35(8):761–762, 1986.

[98] C. Mesterharm. Tracking linear-threshold concepts with Winnow. *Journal of Machine Learning Research*, 4:819–838, 2003.

[99] T. M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

[100] R. J. Mooney. A preliminary PAC analysis of theory revision. In *Computational Learning Theory and Natural Learning Systems, volume III: Selecting Good Models*, pages 43–53. MIT Press, 1995.

[101] D. Mubayi, Gy. Turán, and Y. Zhao. The DNF exception problem. *Theoretical Compututer Science*, 352:85–96, 2006.

[102] D. Mundici. Functions computed by monotone Boolean formulas with no repeated variables. *Theoretical Compututer Science*, 66:113–114, 1989.

[103] D. Ourston and R. J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66(2):273–309, 1994.

[104] K. Pillaipakkamnatt and V. Raghavan. Read-twice DNF formulas are properly learnable. *Information and Computation*, 122(2):236–267, 1995. Earlier verion appeared in 1st EuroColt, 1993.

[105] S. Pinker. *The Blank Slate: The Modern Denial of Human Nature*. Viking Press, 2002.

[106] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Jourrnal of the ACM*, 35(4):965–984, 1988.

[107] B. L. Richards and R. J. Mooney. Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

[108] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[109] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison Wesley, 1998.

[110] P. Savický. On determinism versus unambiquous nondeteminism for decision trees. *Electronic Colloquium on Computational Complexity (ECCC)*, Technical Report TR02-009, 2002.

[111] P. Savický and J. Sgall. DNF tautologies with a limited number of occurrences of every variable. *Theoretical Computer Science*, 238:495–498, 2000.

[112] M. Schmitt. On methods to keep learning away from intractability. In *Proc. International Conference on Artifical Neural Networks (ICANN 1995)*, vol. 1, pages 211–216, 1995.

[113] R. H. Sloan and B. Szörényi. Revising projective DNF in the presence of noise. In *Proc. Kalmár Workshop on Logic and Computer Science*, pages 143–152, 2003.

[114] R. H. Sloan, B. Szörényi, and Gy. Turán. On k-term DNF with the maximal number of prime implicants. Accepted for publication at *SIAM Journal on Discrete Mathematics*. Earlier version appeared as Electronic Colloquium on Computational Complexity (ECCC) Technical Report TR05-023.

[115] R. H. Sloan, B. Szörényi, and Gy. Turán. Projective DNF Formulae and Their Revision. Accepted for publication at *Discrete Applied Mathematics*. Earlier version appeared in 16th COLT, 2003.

[116] R. H. Sloan, B. Szörényi, and Gy. Turán. Revising threshold functions. *Theoretical Computer Science*, 382(3):198–208, 2007.

[117] R. H. Sloan and Gy. Turán. Learning from incomplete boundary queries using split graphs and hypergraphs. In *Proc. 3rd European Conference on Computational Learning Theory (EuroCOLT 1997)*, vol. 1208 of LNAI, pages 38–50. Springer, 1997.

[118] R. H. Sloan and Gy. Turán. On theory revision with queries. In *Proc. 12th Conference on Compututational Learning Theory*, pages 41–52. ACM Press, 1999.

[119] B. Szörényi. Disjoint DNF Tautologies with Conflict Bound Two. Accepted for publication at *Journal on Satisfiability, Boolean Modeling and Computation*.

[120] C. Stone. Consistent nonparametric regression. *Annals of Statistics*, 5:595–645, 1977.

[121] G Tardos. Query complexity, or why is it difficult to separate NP^A∩ coNP^A from P^A by random oracles A? *Combinatorica*, 9(4):385–392 1989.

[122] C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

[123] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[124] R. Uehara, K. Tsuchida, and I. Wegener. Identification of partial disjunction, parity, and threshold functions. *Theoretical Computer Science*, 230:131–147, 1999.

[125] A. Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, 1995.

[126] L. G. Valiant. A neuroidal architecture for cognitive computation. *Journal of the ACM*, 47(5):854–882, 2000.

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

[128] L. G. Valiant. Projection learning. *Machine Learning*, 37(2):115–130, 1999.

[129] L. G. Valiant. Robust logics. *Artificial Intelligence*, 117:231–253, 2000.

[130] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Jourrnal of the ACM*, 23:733–742, 1976.

[131] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[132] I. Wegener. *The Complexity of Boolean Functions*. Wiley–Teubner, 1987.

[133] P. H. Winston, T. O. Binford, B. Katz, and M. Lowry. Learning physical descriptions from functional definitions, examples, and precedents. In *Proc. National Conference on Artificial Intelligence*, pages 433–439, 1983.

[134] S. Wrobel. *Concept Formation and Knowledge Revision*. Kluwer, 1994.

[135] S. Wrobel. First order theory refinement. In *Advances in ILP*, pages 14–33. IOS Press, Amsterdam, 1995.