

Classes of Boolean Functions

Nader H. Bshouty

Eyal Kushilevitz

Abstract

Here we give classes of Boolean functions that considered in COLT.

1 Classes of Functions

Here we introduce the basic classes of functions for which we will present algorithms throughout this Lecture Notes. Each of these classes will be defined and some facts about the relationships between these classes will be presented. The chapter does not discuss any issue related to the learnability of these classes; these issues will be discussed in subsequent chapters.

Conventions and Notations: We will use x_1, \dots, x_n to denote our variables, where in particular n is the number of variables. A *literal* is a variables or a negated variable. That is, the set of all literals is $\{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$. Each variables x_i can get either the value TRUE or the value FALSE; its negation \bar{x}_i gets the opposite value. For convenience, we will use most of the time the value 1 to represent TRUE and the value 0 to represent FALSE. The functions that we introduce are mostly boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Given an n -bit binary string $a = a_1 a_2 \dots a_n$ we view it as an assignment for the n variables, where the value for x_i is the bit a_i (either 0 or 1).

2 Disjunctive Normal Form (DNF) Formulae

In this section we define the class DNF of *disjunctive normal form* formulae. The basic elements in such formulae are called *terms*. A term is simply a conjunctions of literals. That is,

$$T = \ell_1 \wedge \ell_2 \wedge \dots \wedge \ell_t,$$

where all ℓ_i 's are literals and \wedge denotes the boolean AND operation. A term is *satisfied* (that is, gets the value TRUE) if and only if each of the literals in the term gets the value TRUE. Therefore, each term is just a function from $\{0, 1\}^n$ to $\{0, 1\}$. We denote by TERMS_n the class of all possible terms over the literals $x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n$. When n is clear from the context we may use TERMS for short. Also, for simplicity of notations we omit the \wedge symbols from the term. That is, we use $x_2 \bar{x}_5 x_9$ to denote $x_2 \wedge \bar{x}_5 \wedge x_9$.

A DNF formulae is a disjunction of terms. That is,

$$F = T_1 \vee T_2 \vee \cdots \vee T_k,$$

where each T_i is a term and \vee denotes the boolean OR operation. The formula is *satisfied* (that is, gets the value TRUE) if and only if at least one of its terms is satisfied. Again, each such formula defines a function from $\{0, 1\}^n$ to $\{0, 1\}$. We denote by k -TERM-DNF $_n$ the class of all DNF formulae with at most k terms, and omit n from the notation when it is clear from the context. We denote by DNF $_n$ (or DNF for short) the class of all DNF formulae whose number of terms is bounded by some polynomial in n . The following is a simple fact about DNF formulae:

Fact 2.1 *Every function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented by a DNF formula with at most 2^n terms.*

A term is called *monotone* if it uses only the variables x_1, \dots, x_n and no negated variable (that is the term $x_2x_5x_9$ is monotone but $x_1\bar{x}_4x_7$ is not a monotone term). We denote by MTERMS the class of monotone terms (that is MTERMS is a subclass of TERMS). A DNF formula is called *monotone* if it uses only monotone terms. For example, $F = x_1x_7x_{12} \vee x_1x_3x_9 \vee x_2x_4x_7x_{11}$ is a monotone DNF formula. We denote by MDNF the subclass of DNF which contains only monotone DNF formulae. A function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is said to be a *monotone function* if for every assignment $a \in \{0, 1\}^n$ if b is an assignment that can be obtained from a by flipping a 0-bit of a to a 1-bit then $f(b) \geq f(a)$; that is, if $f(a)$ is 1 (TRUE) then so is $f(b)$. The following are simple facts about the connections between monotone DNF formulae and monotone functions:

Fact 2.2

1. *Every monotone function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented by a monotone DNF formula with at most 2^n terms. (In fact, $\binom{n}{n/2}$ terms suffice.)*
2. *Every monotone DNF formula represents a monotone function $f : \{0, 1\}^n \rightarrow \{0, 1\}$.*

Exercise 2.1 *Prove Fact 2.1.*

Exercise 2.2 *Prove Fact 2.2.*

Exercise 2.3 *Prove that the class MTERMS contains 2^n functions and that the class TERMS contains 3^n functions.*

3 Conjunctive Normal Form (CNF) Formulae

In this section we define the class CNF of *conjunctive normal form* formulae. CNF formulae are “dual” to DNF formulae in that the \wedge and \vee operators exchange their roles. The basic elements in CNF formulae are called *clauses*. A clause is a disjunction of literals. That is,

$$C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_t.$$

A clause is *satisfied* if and only if at least one of the literals in the clause gets the value TRUE. A CNF formulae is a conjunction of clauses. That is,

$$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m.$$

The formula is *satisfied* if and only if each of its clauses is satisfied. As before, each such formula defines a function from $\{0, 1\}^n$ to $\{0, 1\}$. We denote by CNF_n (or CNF for short) the class of all CNF formulae whose number of clauses is bounded by some polynomial in n . A k -CNF formula is a CNF formula in which each of its clauses contains at most k literals. We denote by k -CNF the subclass of CNF with all k -CNF formulae.

Fact 3.1 For every k ,

$$k\text{-TERM-DNF} \subseteq k\text{-CNF}.$$

Proof: Let $F = T_1 \vee \cdots \vee T_k$ be a k -term DNF formula. Define

$$F' = \bigwedge_{\ell_1 \in T_1, \dots, \ell_k \in T_k} (\ell_1 \vee \cdots \vee \ell_k).$$

(Where $\ell \in T$ means that the literal ℓ appears in the term T .) Clearly, F' is a k -CNF formula. We now argue that F and F' are equivalent. For every assignment a , if $F(a) = 0$ this implies that in every term T_i there is a literal ℓ_i which is not satisfied. Hence the clause $(\ell_1 \vee \cdots \vee \ell_k)$ of F' is not satisfied and so $F'(a) = 0$. Conversely, if $F'(a) = 0$ then there exists a clause $(\ell_1 \vee \cdots \vee \ell_k)$ that is not satisfied, and therefore in every term T_i there is a literal ℓ_i which is not satisfied. Hence $F(a) = 0$ as well. \square

Exercise 3.1 Strengthen Fact 3.1 by proving that for every k and sufficiently large n ,

$$k\text{-TERM-DNF} \neq k\text{-CNF}.$$

4 Decision Trees

A *decision tree* is a binary tree with the following properties: each internal node of the tree is labeled by a literal from $x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n$; from each internal node there are two edges going to the children of this node one is labeled by 0 and the other is labeled by 1; and each leaf of the tree is labeled by either 0 or 1. A decision tree computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in the following way: given an assignment to the n variables, we start at the root of the tree; whenever we reach a node labeled by some literal ℓ we consider the value of ℓ in the assignment (0 or 1) and we proceed by going on the edge which is labeled by this value. When we reach one of the leaves (labeled 0 or 1) we take this label as the value of f on the assignment. For example, in Figure 1 a decision tree is presented which computes a function $f : \{0, 1\}^4 \rightarrow \{0, 1\}$; the value of f on the assignment 0101 is 0.

The *size* of a decision tree is measured by its number of leaves (for example, the decision tree in Figure 1 is of size 6). The *depth* of a node is its distance (in edges) from the root of the

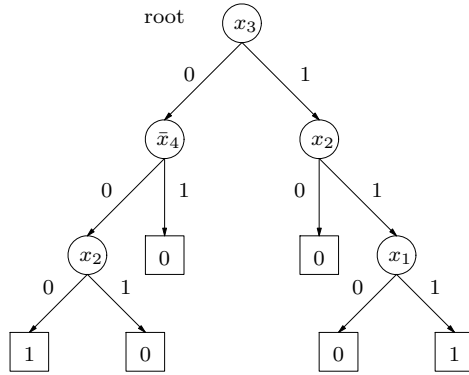


Figure 1: A Decision Tree

tree, and the depth a tree is the maximal depth of a node (or actually of a leaf). We denote by DT_n (or DT for short if n is clear from the context) the class of all functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that can be represented by a decision tree of size polynomial in n . The following fact relates the class DT with the class DNF .

Fact 4.1

$$DT_n \subseteq DNF_n.$$

Proof: Given a decision tree D we show how to construct a DNF formula F that computes the same function. Moreover, if D is of size t then F will have at most t terms. The construction is as follows: for every leaf v of the decision tree D which is labeled by 1 we construct a term T_v for the DNF. The term T_v is constructed by considering the path from the root of D to the leaf v ; for each internal node of the path which is labeled by a literal ℓ , if the edge of the path from this node is labeled 1 we put ℓ in T_v and if this edge is labeled 0 we put $\bar{\ell}$. For example, for the decision tree in Figure 1 we get the following 2-term DNF (corresponding to the 2 nodes in the tree which are labeled by 1):

$$\bar{x}_3 x_4 \bar{x}_2 \vee x_3 x_2 x_1.$$

For the equivalence of F and D note that if D gives the value 1 to some assignment $a \in \{0, 1\}^n$ this implies that when evaluating $D(a)$ we go from the root to some leaf v labeled 1 (over the path connecting the root to v). It follows that the term T_v is satisfied and so is F . Conversely if F is satisfied by some assignment a this implies that some term T_v of F is satisfied. It follows that when evaluating $D(a)$ we will go on the path from the root to the node v and hence the value is v 's label which is 1. \square

Exercise 4.1 An alternative size measure for decision trees is the total number of nodes in a tree (that is, both internal nodes and leaves). Prove that for every decision tree, if m is the number of nodes and t is the number of leaves then $m = 2t - 1$.

Exercise 4.2 Prove that $DT_n \subseteq CNF_n$.

Exercise 4.3 Denote by DISJ-DNF the sub-class of DNF that contains all the functions that can be represented by a DNF formulae whose terms are “disjoint”; that is, every assignment satisfies at most one term. Prove that $DT_n \subseteq DISJ-DNF_n$.

Exercise 4.4 Show that there are polynomial size DNF formulae that require exponential size decision trees.

4.1 Decision Lists

A *decision list* is a restricted type of decision tree. It is defined by t literals ℓ_i and $t + 1$ binary values b_i :

$$(\ell_1, b_1), (\ell_2, b_2), \dots, (\ell_t, b_t), b_{t+1}.$$

Pictorially, it looks as a “single-path” decision tree (see Figure 2). Alternatively, it can be thought of as a generalized if-then-else statement. That is:

```

If  $\ell_1$  Then  $b_1$ 
Else If  $\ell_2$  Then  $b_2$ 
Else If  $\ell_3$  Then  $b_3$ 
    ⋮
Else If  $\ell_t$  Then  $b_t$ 
Else  $b_{t+1}$ 

```

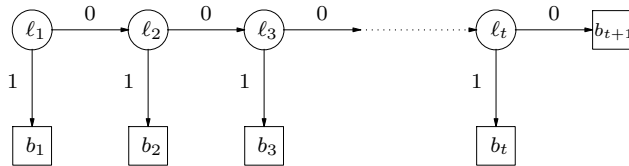


Figure 2: A Decision List

The class DL_n (or DL for short) consists of all functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that can be represented as decision lists. A natural extension of decision lists allows using terms in the nodes instead of literals. Such a list is defined by t terms T_i and $t + 1$ binary values b_i :

$$(T_1, b_1), (T_2, b_2), \dots, (T_t, b_t), b_{t+1}.$$

In particular, we will call k -DL the class of all functions that can be represented by a decision list (not necessarily of polynomial size) in which the nodes contain terms with at most k literals and where the *length* of the list, t , is polynomial in n . For instance, 1-DL is the same as DL.

Exercise 4.5 Prove that $\text{TERMS} \subseteq \text{DL}$.

Exercise 4.6 Let k -DNF be the class of all DNF formulae where each of the terms contains at most k literals. Prove that k -DNF $\subseteq k$ -DL.

Exercise 4.7 Prove that for every k and sufficiently large n , there exists a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ such that $f \in k$ -DL but $f \notin k$ -DNF.

Exercise 4.8 Prove that $\text{DL} \subseteq n$ -TERM-DNF.

Exercise 4.9 Prove that the class DL contains at most $(n + 1)! \cdot 4^n$ functions and at least $n!$ functions.

4.2 *Decision Lists vs. Other Representations

Below we give some more sophisticated relationships between decision lists and other classes; specifically, decision trees and DNF formulae.

Fact 4.2

$$\text{DT} \subseteq O(\log n)\text{-DL}.$$

Moreover, the decision list corresponding to a polynomial size decision tree is also of polynomial size.

Proof: Given a decision tree D of size s we will show how to construct a decision list L in $\log s$ -DL. Moreover, the length of L is at most s . Notice that since D is of size s it must contain a leaf v in depth $\log s$. Let T_v be the term corresponding to v (the one that defines the set of assignments that reach v , as in the proof of Fact 4.1) and let b_v be the label of the leaf v . Let D' be the tree obtained from D by removing v and shrinking the tree (that is, removing v 's father and pointing to v 's brother instead). Observe that the size of D' is smaller than the size of D by at least one. The list L will start with (T_v, b_v) and continues by recursively applying the same process to D' (once we get a tree that computes a constant value b we add it as a last item in the decision list and stop). It can be readily verified that L is indeed of length at most s , each of its nodes is a term of at most $\log s$ literals and that L and D compute the same function. \square

Fact 4.3

$$\text{DNF} \subseteq O(\sqrt{n} \log n)\text{-DL}.$$

Proof: Given a DNF formula $F = T_1 \vee \cdots \vee T_s$ over $r \leq n$ variables, we say that a term T_i is *long* if it has more than $\sqrt{n} \log n$ literals. Let $t \leq s$ be the number of long terms in F . Consider these t long terms. Since each of them contains more than $\sqrt{n} \log n$ literals then there are at least $t\sqrt{n} \log n$ literals overall in these terms and so (at least) one of the $2n$ literals $x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n$ appears at least $\frac{t\sqrt{n} \log n}{2n} = \frac{t \log n}{2\sqrt{n}}$ times; that is, in at least $\frac{t \log n}{2\sqrt{n}}$ terms. Let ℓ be such a literal. We construct a decision tree of a special form using this literal. The root of the tree contains the literal ℓ ; the edge which is labeled 0 points to a node labeled with a DNF formula F_0 which is the DNF that results from F by substituting 0 in ℓ ; that is, remove from F every term that contains the literal ℓ and omit all appearances of the literal $\bar{\ell}$ from the other terms. Similarly, the edge which is labeled 1 points to a node labeled with the DNF formula F_1 that results from F by substituting 1 in ℓ . Since F contains $\frac{t \log n}{2\sqrt{n}}$ terms that contains ℓ , the number of long terms in F_0 is at most $t - \frac{t \log n}{2\sqrt{n}}$. Note that both F_0 and F_1 are functions of (at most) $r - 1$ variables and that the function computed by the above decision tree is identical to F .

Now, recursively apply the above procedure to each of F_0 and F_1 where the recursion is stopped at any node where none of the terms is long. Denote by $\Phi(r, t)$ the size of the largest decision tree that can be obtained by the above procedure from a DNF formula with at most r variables and at most t long terms (note that this is not a regular decision tree since its leaves are not labeled by constants but rather contain DNF formulae with “short” terms). By the construction we get that

$$\Phi(r, t) \leq \Phi\left(r - 1, t - \frac{t \log n}{2\sqrt{n}}\right) + \Phi(r - 1, t).$$

By applying the recursion to the second summand many times we get

$$\begin{aligned} \Phi(n, t) &\leq \Phi\left(r - 1, t - \frac{t \log n}{2\sqrt{n}}\right) + \Phi\left(r - 2, t - \frac{t \log n}{2\sqrt{n}}\right) + \cdots + \Phi\left(1, t - \frac{t \log n}{2\sqrt{n}}\right) \\ &\leq r \cdot \Phi\left(r - 1, t - \frac{t \log n}{2\sqrt{n}}\right). \end{aligned}$$

Again, applying this recursion k times for $r = n$ we get

$$\Phi(n, t) \leq n^k \cdot \Phi\left(n - 1, t \left(1 - \frac{\log n}{2\sqrt{n}}\right)^k\right).$$

For $k = 2\sqrt{n} \ln t / \log n$ we have that $t \left(1 - \frac{\log n}{2\sqrt{n}}\right)^k < 1$ and so the right multiplicand becomes 1 and hence $\Phi(n, t) \leq n^{2\sqrt{n} \ln t / \log n}$. Since $t \leq s = \text{poly}(n)$, we get a decision tree of size $\Delta \leq n^{2\sqrt{n} \ln t / \log n} = n^{O(\sqrt{n})}$ whose leaves are labeled by DNF formulae with terms of length at most $\sqrt{n} \log n$. We now apply the proof of Fact 4.2 to these trees. That is, we construct the decision list by picking a leaf v of depth at most $\log \Delta = O(\sqrt{n} \log n)$. If T_v is the term corresponding to the path to v and $T_{v,1} \vee \cdots \vee T_{v,m}$ is the DNF formula that appears in the node v then we add to the decision list the items

$$(T_v \wedge T_{v,1}, 1), \dots, (T_v \wedge T_{v,m}, 1), (T_v, 0).$$

The term that appears in each item is of the desired length. We leave the proof of correctness for the reader. \square

Exercise 4.10 Prove that every decision tree of size s contains a leaf v in depth $\log s$.

Exercise 4.11 Verify the correctness of the decision list L constructed in the proof of Fact 4.2.

Exercise 4.12 Verify the correctness of the decision list L constructed in the proof of Fact 4.3.

Exercise 4.13 Prove that any n -DL of length s is subset of $O(\sqrt{n \log n \log s})$ -DL.

5 Deterministic Automata

In this section we define deterministic (finite) automata. In particular, we focus on viewing these automata as a tool for computing boolean functions on n boolean variables.

A *deterministic finite automaton* (DFA for short) \mathcal{A} has a finite set of states, Q , and a transition function $\delta : Q \times \Sigma \rightarrow Q$ which determines for each state and for each character in the alphabet Σ what should be the next state. Every input $w = w_1 w_2 \dots w_n$ defines a sequence of $(n + 1)$ states as follows: start with the *initial state* (a pre-specified state $q_0 \in Q$), and in the i -th step according to the current state q and the character w_i move to the state defined by δ (that is, to the state $\delta(q, w_i)$). If at the end the state of the automaton belongs to the set of *accepting states* (a pre-specified set $F \subseteq Q$) then we say that w is accepted by \mathcal{A} or that w belongs to the *language* of \mathcal{A} . Otherwise, we say that w is rejected by \mathcal{A} or that w does not belong to the language of \mathcal{A} . We denote by DFA_m the collection of all languages that can be accepted by a DFA with at most m states.

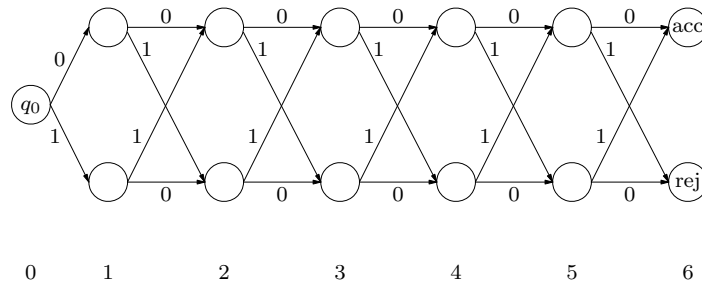


Figure 3: A Boolean deterministic finite automaton

Next, we wish to put some additional restrictions on the automata. We view an automaton \mathcal{A} as computing a boolean function f which is defined as $f(w) = 1$ if and only if \mathcal{A} accepts the string w . Since we are interested in functions defined over the domain $\{0, 1\}^n$ we will restrict our alphabet, Σ to $\{0, 1\}$. Moreover, to make sure our automata accept only strings of length

n we will consider n -level automata. That is, the set of states Q is partitioned into levels where the initial state, q_0 , is in level 0; the accepting states are all in level n ; and from each state q in level i all possible transitions lead to level $i + 1$ (that is, for every $b \in \{0, 1\}$ the state $\delta(q, b)$ is in level $i + 1$). We call such a DFA a *boolean DFA* (or B DFA, for short); See Figure 3 for an example of a 6-level automaton. The following is an easy fact about BDFAs:

Fact 5.1 *Let \mathcal{A} be a B DFA. Then, all the strings accepted by \mathcal{A} are of length exactly n .*

By the above fact, we can think of every B DFA as computing a boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where as usual every n -bit string can be viewed as an assignment to the n variables. For example, the B DFA in Figure 3 gives the value 1 (that is, accepts) all the strings of length 6 with an even number of 1s. We denote by B DFA_n the collection of all such functions that can be computed by an n -level B DFA whose number of states is polynomial in n (as usual, when n is clear from the context we omit it from the notation).

Exercise 5.1 *Prove Fact 5.1.*

Exercise 5.2 *Prove that $O(\log n)$ -TERM-DNF $\subseteq \text{B DFA}_n$. More precisely, show that for every k -term DNF there exists an equivalent B DFA with at most $2^k \cdot n$ states.*

Exercise 5.3 *Show that for every function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that depends only on k (out of the n) variables there exists an equivalent B DFA with at most $2^k \cdot n$ states.*

Exercise 5.4 *Prove that if a language L is in DFA_s then $L \cap \{0, 1\}^n$ is in B DFA_{ns} .*

Exercise 5.5 *Define the width of a B DFA to be the maximum over all the levels of the number of nodes in a level. Show that if g_1, g_2, \dots, g_k have B DFA of width d_1, \dots, d_k then for any function f , the function $f(g_1, \dots, g_k)$ have a B DFA of width $d_1 d_2 \dots d_k$. Use this to prove exercise 5.3.*

6 The Hankel Matrix Representation

In this section we present the *Hankel matrix representation* of functions. While this representation may be seemed un-natural in first sight, we will show in this section that it contains several important subclasses; later in the book we will show that this representation has interesting applications in learning various classes of functions. These applications build upon the algebraic nature of the representation.

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a function. The *Hankel matrix* corresponding to f , denoted F , is defined as follows: each row of F is indexed by a string $x \in \{0, 1\}^{\leq n}$; each column of F is indexed by a string $y \in \{0, 1\}^{\leq n}$; the (x, y) entry of F contains the value of $f(x \circ y)$ (where \circ denotes concatenation) if $x \circ y$ is of length (exactly) n and the value 0 otherwise. We denote by

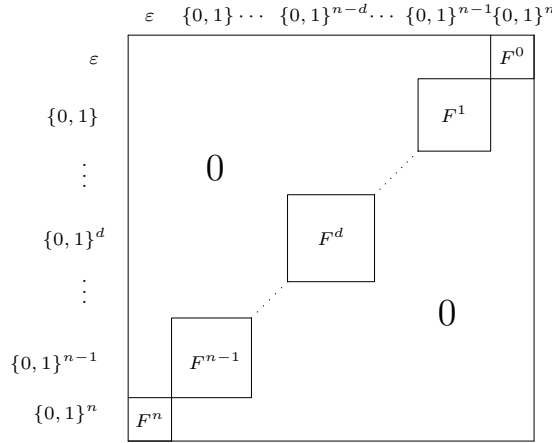


Figure 4: The Hankel matrix F

F^d the submatrix of F whose rows are indexed by strings $x \in \{0,1\}^d$ and whose columns are indexed by strings $y \in \{0,1\}^{n-d}$ (see Fig. 4).

The complexity measure of a function f represented by its Hankel matrix, F , is defined to be $\text{rank}(F)$ (that is, the number of independent rows/columns in F). We define HANKEL_n to be the set of all functions $f : \{0,1\}^n \rightarrow \{0,1\}$ whose Hankel matrix F is of rank polynomial in n . By the special structure of the matrix and simple linear algebra,

$$\text{rank}(F) = \sum_{d=0}^n \text{rank}(F^d). \quad (1)$$

The following facts show the richness of the class HANKEL .

Fact 6.1

$$\text{BDFAN} \subseteq \text{HANKEL}_n.$$

Proof: Let f be a function in BDFAN and let \mathcal{A} be the corresponding n -level BDFA that computes the function f . Denote by m the number of states in \mathcal{A} . We will show that each of the submatrices F^d has rank of at most m . By Equation (1) it follows that $\text{rank}(F) \leq m \cdot (n+1)$.

Denote by Q_d the set of all states in level d of \mathcal{A} . For $q \in Q_d$ let M_q be a 0-1 matrix whose rows are indexed by strings $x \in \{0,1\}^d$, whose columns are indexed by strings $y \in \{0,1\}^{n-d}$, and whose (x,y) entry contains 1 if and only if on the string x the automaton \mathcal{A} reaches from the initial state to the state q and on the string y the automaton \mathcal{A} reaches from state q to an accepting state (otherwise the (x,y) entry is 0). Observe that

$$F^d = \sum_{q \in Q_d} M_q \quad (2)$$

and that the rank of each matrix M_q is exactly 1. By linear algebra $\text{rank}(F^d) \leq |Q_d| \leq m$, as desired. \square

Fact 6.2

$$\text{DT}_n \subseteq \text{HANKEL}_n.$$

Proof: Let D be a decision-tree of size m . As in the proof of Fact 6.1, we will show that each of the submatrices F^d has rank of at most m . By Equation (1) it follows that $\text{rank}(F) \leq m(n+1)$.

Fix some value of d , and define for every leaf v of the tree D a 0-1 matrix M_v whose rows are indexed by strings $x \in \{0,1\}^d$, whose columns are indexed by strings $y \in \{0,1\}^{n-d}$, and whose (x,y) entry contains 1 if and only if the string $x \circ y$ reaches the leaf v of D (otherwise the (x,y) entry is 0). Observe that

$$F^d = \sum_v M_v \tag{3}$$

and that the rank of each matrix M_v is exactly 1. By linear algebra $\text{rank}(F^d) \leq m$, as desired. \square

Exercise 6.1 Prove Equation (2) and that the rank of each matrix M_q is exactly 1.

Exercise 6.2 Prove Equation (3) and that the rank of each matrix M_v is exactly 1.

Exercise 6.3 Recall the definition of the class DISJ-DNF (Exercise 4.3). Prove that $\text{DISJ-DNF}_n \subseteq \text{HANKEL}_n$.