Recognition of Interval Boolean Functions

Ondřej Čepek[a] David Kronus[b] Petr Kučera[c]

RRR 15-2006, JUNE, 2006

[a]Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Prague, <ondrej.cepek@mff.cuni.cz>
[b]Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Prague, <david.kronus@mff.cuni.cz>
[c]Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Prague, <petr.kucera@mff.cuni.cz>

Rutcor Research Report
RRR 15-2006, June, 2006

Recognition of Interval Boolean Functions

Ondřej Čepek David Kronus Petr Kučera

Abstract. Interval functions constitute quite a special class of Boolean functions for which it is very easy and fast to determine their functional value on a specified input vector. The value of an n-variable interval function specified by interval $[a, b]$ (where a and b are n-bit binary numbers) is *true* if and only if the input vector viewed as an n-bit number belongs to the interval $[a, b]$. In this paper we study the problem of deciding whether a given DNF represents an interval function and if so then we also want to output the corresponding interval. For general Boolean functions this problem is co-NP-hard. In our article we present a polynomial time algorithm which works for monotone functions. We shall also show that given a Boolean function f belonging to some class $\mathcal{C}$ which is closed under partial assignment and for which we are able to solve the satisfiability problem in polynomial time, we can also decide whether f is an interval function in polynomial time. We show how to recognize a "renamable" variant of interval functions, i.e. their variable complementation closure. Another studied problem is the problem of finding an interval extension of partially defined Boolean functions. We also study some other properties of interval functions.

Acknowledgements: This research was supported by Czech Science Foundation (GACR), grant number 201/05/H014.

1 Introduction

The class of interval functions was introduced in [9], where the following problem was presented: Given two n-bit numbers a, b, find a minimum (shortest) DNF representing a Boolean function f on n variables, which is true exactly on numbers from the interval $[a, b]$. This problem originated from the field of automatic generation of test patterns for hardware verification, see e.g. [7, 6]. In [9] the authors also studied the problem in which we are looking for a shortest DNF $\mathcal{F}$ representing f (given by an interval [a,b]) such that for any vector x, for which $f(x) = 1$ there is exactly one term t in $\mathcal{F}$ for which $t(x) = 1$. In other words, there is an additional requirement that the sets on which the terms of $\mathcal{F}$ are satisfied must be pairwise disjoint.

The representation of an interval Boolean function using only the two n-bit numbers a, b can be rather useful (and in particular it is very short). Despite this fact, we consider in this paper the reverse problem which also appears to be interesting and practically important: Given a DNF $\mathcal{F}$, can we recognize whether it represents an interval function? This problem is co-NP hard in general, as we shall show in Section 3. Moreover, this problem remains co-NP hard even if we fix in advance the order of variables according to their significancy (when we view a Boolean vector as an n-bit binary number), and test whether the input DNF represents an interval function with respect to this fixed order. Fortunately, even the general problem (without a fixed order of variables) is polynomially solvable when we restrict our attention only to those classes of DNFs, for which we are able to solve the satisfiability problem in polynomial time. The most trivial example is the class of monotone functions.

As we shall see in Section 7, the class of interval functions is not closed under variable complementation, hence we also introduce a complementation closure of interval functions and describe, how to recognize, whether a given DNF represents a function which belongs to a complementation closure of interval functions. Our algorithm is polynomial under the same assumptions as the above mentioned recognition algorithm for interval DNFs.

Another widely studied and practically important problem is finding an extension of a pdBf in a specified class. This problem can be defined as follows: Given partially defined Boolean function (T, F) find a function from a specified class which is true on all vectors from T and false on all vectors from F, or prove that such a function does not exist. In [1] this problem was studied for a variety of classes of Boolean function. For some of the classes the problem is solvable in polynomial time, for others it is NP-hard. In this paper we show, that given a partially defined Boolean function, we can find its interval extension, if it exists, in polynomial time.

1.1 Outline

The paper is structured as follows. In Section 2 we introduce the necessary notation and give basic definitions. The notion of an interval function is also made precise here. In Section 3 we address the general recognition problem for interval functions. We show that it is co-NP hard to recognize whether a given (general) DNF represents an interval function. However, the situation is very different when the input DNF fulfils some additional requirements. First, in Section 4, we show that when the input DNF is positive or negative, it is easy to find out, whether it represents an interval function. This is further extended in Section 5 where we show that given an arbitrary prime and irredundant DNF $\mathcal{F}$, it is possible to check in polynomial time, whether $\mathcal{F}$ represents an interval function (note that any positive or negative DNF can be trivially transformed into a prime and irredundant one by performing absorbtions). As a consequence of this result we get, that given a class $\mathcal{C}$ of DNFs for which we are able to

1. solve the satisfiability problem in polynomial time for every DNF $\mathcal{F}$ in $\mathcal{C}$ and

2. which is closed under partial assignment

then we can test, whether $\mathcal{F}$ represents an interval function for every DNF $\mathcal{F}$ in $\mathcal{C}$ (even if $\mathcal{F}$ is not prime and irredundant). The reason is that the above two properties suffice for the existence of a polynomial time procedure which transforms an arbitrary DNF in $\mathcal{C}$ into a logically equivalent DNF which is prime and irredundant.

In Section 6 we derive an interesting structural result for interval functions. We observe, that each interval function f can be written as a conjunction of a positive interval function f_P and a negative interval function f_N. Then we show, that given a prime and irredundant DNF $\mathcal{F}$ representing f, we can easily find a positive DNF $\mathcal{F}^P \leq f_P$ and a negative DNF $\mathcal{F}^N \leq f_N$, such that $\mathcal{F}^P \wedge \mathcal{F}^N$ represents f. In Section 7 we study the variable complementation closure of the class of interval functions. Here we show, that given a prime and irredundant DNF $\mathcal{F}$, we can test in polynomial time, whether $\mathcal{F}$ belongs to the variable complementation closure of interval functions, i.e. whether $\mathcal{F}$ represents a function obtainable from some interval function by complementing a subset of its variables. Finally, in Section 8 we show, that it is possible to find an interval extension of a pdBf, if it exists. We close our paper with few concluding remarks in Section 9.

2 Notation and definitions

A *Boolean function*, or a *function* in short, is a mapping $f : \{0,1\}^n \mapsto \{0,1\}$, where $x \in \{0,1\}^n$ is called a *Boolean vector* (a *vector* in short). Propositional variables $x_1, \ldots, x_n$ and their negations $\overline{x}_1, \ldots, \overline{x}_n$ are called *literals* (*positive* and *negative literals* respectively). An elementary conjunction of literals

$$t = \bigwedge_{i \in I} x_i \wedge \bigwedge_{j \in J} \overline{x}_j \tag{1}$$

is called a *term*, if every propositional variable appears in it at most once, i.e. if $I \cap J = \emptyset$. A *disjunctive normal form* (or DNF) is a disjunction of terms. It is a well known fact (see e.g. [4].), that every Boolean function can be represented by a DNF. For a DNF $\mathcal{F}$ and a term t we denote by $t \in \mathcal{F}$ the fact, that t is contained in $\mathcal{F}$. In the subsequent text the "$\wedge$" sign will be frequently omitted.

The DNF version of the *satisfiability problem* (sometimes called the *falsifiability problem*) is defined as follows: given a DNF $\mathcal{F}$, does there exist an assignment of truth values to the variables which makes $\mathcal{F}$ evaluate to 0?

Given Boolean functions f and g on the same set of variables, we denote by $f \leq g$ the fact that g is satisfied for any assignment of values to the variables for which f is satisfied. We call a term t an *implicant* of a DNF $\mathcal{F}$, if $t \leq \mathcal{F}$. We call t a *prime implicant*, if t is an implicant of $\mathcal{F}$ and there is no implicant $t' \neq t$ of $\mathcal{F}$, for which $t \leq t' \leq \mathcal{F}$. We call DNF $\mathcal{F}$ prime, if it consists of only prime implicants. We call DNF $\mathcal{F}$ irredundant if for any term $t \in \mathcal{F}$, DNF $\mathcal{F}'$ produced from $\mathcal{F}$ by deleting t does not represent the same function as $\mathcal{F}$. It is a well known fact, that if $\mathcal{F}$ belongs to some class of DNFs $\mathcal{C}$, for which we can solve the satisfiability problem in polynomial time and which is closed under partial assignment, then we can test in polynomial time for a term t and a DNF $\mathcal{F}$, whether t is an implicant of $\mathcal{F}$. To see this, observe that given a term $t = x_1 \ldots x_{l_p} \overline{y}_1 \ldots \overline{y}_{l_n}$, t is an implicant of f if and only if $\mathcal{F}[x_1 := 1, \ldots, x_{l_p} := 1, y_1 := 0, \ldots, y_{l_n} := 0]$ is not falsifiable (there is no assignment to the remaining variables which makes the DNF evaluate to 0).

We say, that two terms t_1 and t_2 *conflict in a variable* x, if t_1 contains literal x and t_2 contains literal $\overline{x}$. Two terms t_1 and t_2 have a *consensus*, if they conflict in exactly one variable. If $t_1 = Ax$ and $t_2 = B\overline{x}$, where A, B are two sets of literals and x is the only variable, in which t_1 and t_2 have conflict, we call a term $t = AB$ a *consensus of* terms t_1 and t_2. We denote this fact by $t = cons(t_1, t_2)$. It is a well known fact (see [8]), that any prime implicant of a DNF $\mathcal{F}$ can be generated from $\mathcal{F}$ by a series of consensuses.

A term is called *positive* if it consists only of positive literals, it is called *negative* if it consists only of negative literals. A DNF $\mathcal{F}$ is called *positive* (*negative* resp.) if it consists only of positive (negative resp.) terms. A function f is called *positive* (*negative* resp.) if it admits a positive (negative resp.) representation. The class of positive functions will be denoted by $\mathcal{C}^+$, the class of negative functions will be denoted by $\mathcal{C}^-$. Throughout the paper we shall use a well known fact that a positive or negative function has the unique prime and irredundant DNF, which is observable using the Quine's theorem ([8]) and a simple fact that in a positive or negative DNF no consensus is possible.

We will denote binary vectors by $\vec{x}, \vec{y}, \ldots$. The bits of vector $\vec{x} \in \{0,1\}^n$ will be denoted by $x_1, \ldots, x_n$. The vector $\vec{x}$ also corresponds to an integer number x with binary representation equal to $\vec{x}$. In this case x_1 is the most significant bit of x and x_n the least significant bit. Hence $x = \sum_{i=1}^{n} x_i 2^{n-i}$.

Definition 2.1 For vector $\vec{x} \in \{0,1\}^n$ and for permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ we denote by $\vec{x}^\pi$ the vector of n-bits formed by permuting bits of $\vec{x}$ by π. That means $x_i^\pi = x_{\pi(i)}$. By x^π we denote the number with binary representation $\vec{x}^\pi$.

Definition 2.2 Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called an *interval function* if there exist two n-bit integers a, b and permutation π of $\{1, \ldots, n\}$ such that for every n-bit vector $\vec{x} \in \{0,1\}^n$ we get $f(\vec{x}) = 1$ if and only if $x^\pi \in [a, b]$. The class of interval functions will be denoted by $\mathcal{C}_{int}$. The class of positive (negative resp.) interval functions will be denoted by C^+_{int} (C^-_{int} resp.).

3 Hardness of the General Recognition Problem

We study the following problem: We are given DNF $\mathcal{F}$ and we want to decide whether $\mathcal{F}$ represents an interval function and if so we also want to output the permutation of input bits and the interval $[a, b]$ which prove that $\mathcal{F}$ represents an interval function. This problem is hard when we consider general DNFs (i.e. with no additional requirements) as inputs:

Theorem 3.1 *It is co-NP-hard to decide whether a given general DNF represents an interval function. This remains true even when we are given a permutation π specifying the only permutation of input bits we should consider. In the latter case the problem is co-NP-complete.*

Proof : Let DNF $\mathcal{F}$ be given. Let us first prove the co-NP-hardness. We will take an instance of the problem TAUT which is defined as follows: given a DNF $\mathcal{F}$ decide whether $\mathcal{F}$ is a tautology or, in other words, if F evaluates to 1 for all assignments of truth values to variables. This problem is known to be co-NP-complete [3]. By deciding whether $\mathcal{F}$ represents an interval function (for some ordering of input bits) and determining corresponding interval $[a, b]$ we can provide answer to the problem TAUT because $\mathcal{F}$ represents tautology if and only if $[a, b]$ equals $[0, 2^n - 1]$ that is $[a, b]$ spans all the integers represented by vectors from $\{0, 1\}^n$. Moreover, for tautology it is not significant which order of variables we consider, tautology is constant 1 on the whole interval $[0, 2^n - 1]$ for every order.

To see that the latter version of our problem is in co-NP we only need to observe that when we are given the permutation π of variables and three binary vectors x, y, z such that $x^\pi < y^\pi < z^\pi$ and $\mathcal{F}(\vec{x}) = \mathcal{F}(\vec{z}) = 1$ and $\mathcal{F}(\vec{y}) = 0$ we have a certificate of a negative answer which can be verified in polynomial time. ■

4 Positive and Negative Interval Boolean Functions

In this section we shall at first develop an algorithm recognizing positive interval functions and then we will show how to use it to recognize negative interval functions.

It is easy to observe that any positive interval function is defined by interval $[a, 2^n - 1]$ for some a (or it is constantly zero). This is because if there is any vector $\vec{x} \in \{0, 1\}^n$ for which $f(\vec{x}) = 1$ then this must also hold for vector $\vec{y} = 1^n$ according to the definition of a positive function. And such $\vec{y}$ represents the number $2^n - 1$ (in any permutation of input bits). For similar reasons, negative interval functions represent intervals of type $[0, b]$ for some b.

Also any interval function defined by interval of type $[a, 2^n - 1]$ is positive. This is because by flipping some zeros to ones in any $x \in [a, 2^n - 1]$ the resulting integer remains in this interval. The following lemma states the condition that makes it possible to effectively decide whether a given positive function is an interval function:

Lemma 4.1 *Let f be a positive interval Boolean function with respect to permutation π of variables which represents interval $[a, 2^n - 1]$, where $a > 0$. Let $c = a - 1 = c^1 c^2 \ldots c^n$ and let x_i be the most significant variable with respect to π (that is $i = \pi^{-1}(1)$). Let $\mathcal{F}$ be a prime*

DNF representation of f. Then either x_i is a linear term in $\mathcal{F}$ or x_i appears in every term of $\mathcal{F}$. Moreover $f' = f[x_i := c^1]$ is again an interval function representing interval $[a', 2^{n-1} - 1]$, where $a' - 1 = c^2c^3 \dots c^n$, and $\mathcal{F}[x_i := c^1]$ is a prime DNF representation of f'.

Proof : As we mentioned above, any positive interval function represents (possibly empty) interval $[a, 2^n - 1]$ for some a. In the case $a \leq 2^{n-1}$ for any vector $\vec{z} \in \{0,1\}^n$ for which the most significant bit of z^π equals 1 it is necessary for $f(\vec{z})$ to be 1 because $z^\pi \geq 2^{n-1} \geq a$. Hence in $\mathcal{F}$ there must be the positive linear term formed by variable x_i corresponding to this most significant bit otherwise $\mathcal{F}$ is not prime. By fixing x_i to $c^1 = 0$ in $\mathcal{F}$ we just remove x_i from it. The resulting $\mathcal{F}'$ represents function f' and behaves identically as f on vectors with x_i equal to 0. Thus it is an interval function representing $[0c^2 \dots c^n + 1, 2^{n-1} - 1] = [c^2 \dots c^n + 1, 2^{n-1} - 1]$. DNF $\mathcal{F}'$ is again prime and irredundant because we only removed linear term x_i and x_i doesn't occur in any other term of $\mathcal{F}'$.

In the case $a > 2^{n-1}$ function f must be 0 on any vector $\vec{z} \in \{0,1\}^n$ for which z^π has the most significant bit set to 0 because all these vectors represent numbers smaller than a. Therefore by fixing variable x_i to 0 in $\mathcal{F}$ we must get DNF equal to constant zero. That means that every term of $\mathcal{F}$ must contain the positive literal x_i. By setting x_i to $c^1 = 1$ in $\mathcal{F}$ we get DNF $\mathcal{F}'$ where every term has x_i removed from it. DNF $\mathcal{F}'$ represents function f' which behaves identically as f on vectors with x_i equal to 1. Thus it is again interval function representing interval formed from $[1c^2 \dots c^n + 1, 2^n - 1]$ by removing the first bit of both boundaries, that is $[c^2 \dots c^n + 1, 2^{n-1} - 1]$. DNF $\mathcal{F}'$ is again prime and irredundant based on the same properties of $\mathcal{F}$. ■

From the proof of the previous lemma it can be seen that the condition stated in the lemma is also sufficient for a positive DNF to represent an interval function. This suggests an algorithm for deciding whether a given positive DNF represents an interval function. In every iteration of the algorithm we will look for one variable satisfying the properties of Lemma 4.1 (i.e. a variable appearing as a linear term or contained in every term) and remove it by fixing its value.

Algorithm 4.2 Positive Interval Function Recognition

Input: *A nonempty prime and irredundant positive DNF $\mathcal{F}$ on n variables representing function f.*

Output: *Order $x_1, \dots, x_n$ of variables, and n-bit number a, if $\mathcal{F}$ represents a positive interval function with respect to this order defined by interval $[a, 2^n - 1]$.* **NO** *otherwise.*

```
1: if $\mathcal{F}(1^n) = 0$ then $\mathcal{F}$ is constant 0, output empty interval endif
2: if $\mathcal{F} = 1$ then $\mathcal{F}$ is constant 1, tautology, output $[0, 2^n - 1]$ endif
3: $c := 0$ {$c$ will denote the actual value of $a - 1$}
4: $b := 0$, $m := 0$
5: while $\mathcal{F}$ can be changed by at least one of the following steps do
```

```
 6:     for every variable $y$ constituting linear term of $\mathcal{F}$
 7:     do
 8:        $x_{m+1} := y$, $m := m + 1$
 9:        $\mathcal{F} := \mathcal{F}[y := 0]$
10:     enddo
11:     for every variable $y$ that is contained in every term of $\mathcal{F}$
12:     do
13:        $c := c + 2^{n-1-m}$, $x_{m+1} := y$, $m := m + 1$
14:        $\mathcal{F} := \mathcal{F}[y := 1]$
15:     enddo
16:  enddo
17:  while $m < n$ do
18:  do
19:     $c := c + 2^{n-1-m}$
20:     $m := m + 1$
21:  enddo
22:  if $\mathcal{F} = \emptyset$
23:  then
24:     $a := c + 1$
25:     output $[a, 2^n - 1]$
26:  else
27:     output $NO$.
```

Theorem 4.3 *Algorithm 4.2 correctly recognizes positive prime DNFs representing positive interval functions and for each such function outputs the order of variables and the interval represented by this interval function.*

Proof : The correctness follows from Lemma 4.1. When we process one variable y and fix its value we get another DNF which is again prime and irredundant according to Lemma 4.1 and represents a positive interval function if and only if the original function was a positive interval function. We also set the most significant bit of the boundary c according to the condition that y fulfilled. Now we can iterate the process because all preconditions of algorithm are satisfied.

If we cannot process all variables of DNF $\mathcal{F}$ before halting we know the condition of Lemma 4.1 is not satisfied and thus $\mathcal{F}$ does not represent interval function.

If we can process all variables of $\mathcal{F}$ then it means that a sufficient condition for $\mathcal{F}$ to represent interval function was satisfied but we still need to add 1 bit to c for every variable of f not present in $\mathcal{F}$.

We also note that the order of steps 6 and 11 must be exactly as in our algorithm because in the last iteration of the while loop 5-16 when we always have DNF consisting of just one variable it is necessary to treat it like a linear term and not increase c by one (on line 13) to get the right value of a. ■

Next we will show how to implement this algorithm effectively.

Theorem 4.4 *It is possible to implement Algorithm 4.2 to have time complexity $\Theta(l)$ where l is the number of literals in the input positive DNF $\mathcal{F}$.*

Proof : If we analyze the algorithm we see that it involves at most n iterations. The complexity of every iteration depends on the implementation of the following operations (remaining operations are of constant time complexity because to add 2^k to a we just set one bit to 1):

1. "Find and remove from $\mathcal{F}$ (some) variable that occurs in every term of $\mathcal{F}$"
2. "Find and remove from $\mathcal{F}$ (some) linear term y"

We need to implement these operations in such a way that their running time over all iterations of the algorithm is $O(l)$. We introduce three types of data structures to achieve this. We define a structure $T(t)$ corresponding to every term t, a structure $V(x)$ for every variable x, and also a structure $L(r)$ for every literal r of DNF $\mathcal{F}$. For every term t we will also remember in $n(t)$ the number of variables in t and for every variable x we will store in $t(x)$ the number of literals formed by x (this equals to the number of terms x is in).

In the initialization step we sort terms by $n(t)$ in ascending order and variables by $t(x)$ in descending order. If we employ RadixSort we can achieve this in time $O(l)$ because we are sorting integers in range $[1, l]$. In these orders we put terms in double-linked list $Terms$ and variables in double-linked list $Vars$. Then for every variable x we create a double-linked list of all structures corresponding to literals formed by x. And for every term t we also create a double-linked list of structures corresponding to literals found in t. Every structure $L(r)$ will also have a pointer to the structure corresponding to the variable forming r and to the structure of the term containing r. It is quite clear that we can do this initialization of data structures in time $O(l)$.

Then when we are asked to find some variable x with occurences in all terms we just need to look at the head x of list $Vars$ and check whether the number $t(x)$ equals the number of terms of $\mathcal{F}$. If it is equal we can remove x from $Vars$ in constant time and then we go through the list of literals formed by x and for every such literal r we decrease by one the number $n(t)$ of term t containing r and we also remove r from the list of literals of t. During the whole execution of algorithm we will deal with every literal just once hence we can achieve all these operations in time $O(l)$.

In case we need to find a linear term we just look at the first term t of list $Terms$ and check whether $n(t)$ equals one. If so we can remove t from $Terms$ in constant time and we also need to remove from $Vars$ the variable forming the (sole) literal of t. By using the pointer from literal to its variable we can do this in constant time. This operation hence takes just constant time and as it is called at most n times it fits in time $O(l)$.

Our last note on the implementation is that by removing linear term or variable contained in every term our data structure retains all its properties that it has after initialization, namely the ascending order of terms with respect to $n(t)$ is preserved because either we remove the first term with $n(t)$ equal to one or we remove variable with occurences in all terms and that means we decrease $n(t)$ of all terms. Also the descending order of variables with respect to $t(x)$ is preserved because we only remove first variable or a variable which has $t(x)$ equal to one and so it is among the last variables in the list. Hence we can iterate these operations and our algorithm with this implementation has $\Theta(l)$ time complexity. ∎

Now we will show how to use Algorithm 4.2 for recognition of negative interval functions. Although we could reformulate all the ideas of this section for negative functions, the easiest way to explain the idea is to note that by switching all literals of a positive prime DNF $\mathcal{F}$ representing positive interval function f from positive form to negative we get a DNF representation of a negative interval function f'. And if f represents interval $[a, 2^n - 1]$ then f' represents "mirrored" interval $[0, \bar{a}]$ where $\bar{a}$ is formed from a by exchanging zeros and ones. Hence when we get the task to recognize whether given negative DNF $\mathcal{F}$ represents a negative interval function we can just switch all literals of $\mathcal{F}$ to positive form to get positive DNF $\mathcal{F}'$, then execute Algorithm 4.2 with input $\mathcal{F}'$. If its answer is NO, we also output NO. If the answer is that $\mathcal{F}'$ represents a positive interval function, we also get the permutation and interval and we output the same permutation and the "mirrored" version of the interval.

Corollary 4.5 *It is possible to recognize whether a given prime negative DNF represents a negative interval function in time $\Theta(l)$.*

5 General Interval Functions

In this section we shall prove the following theorem.

Theorem 5.1 *Let $\mathcal{C}$ be a class of DNFs which is closed under partial assignment and for which we can decide the satisfiability problem in polynomial time. Let $\mathcal{F} \in \mathcal{C}$ be a DNF representing Boolean function f, then we can test whether $\mathcal{F} \in \mathcal{C}_{int}$ in polynomial time.*

We shall at first show how to recognize, whether a given prime and irredundant DNF represents an interval function. The proposition of Theorem 5.1 will follow as an easy consequence.

For recognition of interval functions we shall need that interval functions are closed under partial assignment.

Lemma 5.2 *The class of (positive) interval functions is closed under partial assignment.*

Proof : Let f be an interval function with respect to the order of variables $x_1, \dots, x_n$, representing the interval $[a, b]$. Let $f' = f[x_i := v]$, where $1 \leq i \leq n$ and $v \in \{0, 1\}$ are taken arbitrarily. Let us assume by a contradiction that f' is not interval with respect to the order of variables $x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_n$. Then there are three vectors $u'_1, u'_2, u'_3 \in \{0, 1\}^{n-1}$, such that $f'[u'_1] = f'[u'_3] = 1$ while $f'[u'_2] = 0$. But that would mean that there would also be vectors $u_1, u_2, u_3 \in \{0, 1\}^n$ originating from u'_1, u'_2, u'_3 by inserting the bit of value v at the i-th position, for which $f[u_1] = f[u_3] = 1$ while $f[u_2] = 0$, which is a contradiction to the fact that f is an interval function. Since the class of positive functions is closed under partial assignment, the class of positive interval functions is closed under partial assignment as well. ■

Now we are ready to present an algorithm recognizing, whether a given prime and irredundant DNF represents an interval function.

Algorithm 5.3 Interval Function Recognition

Input: *A nonempty prime and irredundant DNF $\mathcal{F} \in \mathcal{C}$ on n variables representing a function f.*
Output: *Order $x_1, \ldots, x_n$ of variables, and n-bit numbers a, b, if $\mathcal{F}$ represents an interval function with respect to this order defined by interval $[a, b]$.* **NO** *otherwise.*

```
1:  $i := 1$
2:  $\mathcal{F}_1 := \mathcal{F}$
3:  while $\mathcal{F}_i \neq \emptyset$ and $i \leq n$ and $(\exists y)\ (\mathcal{F}_i[y := 0] = \emptyset \vee \mathcal{F}_i[y := 1] = \emptyset)$
4:  do
5:    if $\mathcal{F}_i[y := 0] = \emptyset$
6:    then
7:      $a_i := 1$
8:      $b_i := 1$
9:      $\mathcal{F}_{i+1} := \mathcal{F}_i[y := 1]$
10:   else
11:     $a_i := 0$
12:     $b_i := 0$
13:     $\mathcal{F}_{i+1} := \mathcal{F}_i[y := 0]$
14:   endif
15:   $x_i := y$
16:   $i := i + 1$
17: done
18: if $(\exists y)\ (\mathcal{F}_i[y := 0] \in \mathcal{C}^+_{int} \wedge \mathcal{F}_i[y := 1] \in \mathcal{C}^-_{int}$ and both are interval with respect to
    the same order of variables $x_{i+1}, \ldots, x_n$.)
19: then
20:   $x_i := y$
21:   $a_i := 0$
22:   $b_i := 1$
23:   Find interval $[a', 2^{n-i})$ represented by $\mathcal{F}_i[y := 0]$ with respect to order $x_{i+1}, \ldots, x_n$.
      (See Algorithm 4.2)
24:   Find interval $[0, b']$ represented by $\mathcal{F}_i[y := 1]$ with respect to order $x_{i+1}, \ldots, x_n$.
      (See Algorithm 4.2 and Corollary 4.5)
25:   $a := a_1 \ldots a_i a'$
26:   $b := b_1 \ldots b_i b'$
27:   return order of variables $x_1, \ldots, x_n$ and the values of $a, b$.
```

```
28: else
29:   return NO
30: endif
```

Before we prove the correctness of Algorithm 5.3, we shall give some insight in how the algorithm works. Let us consider a DNF $\mathcal{F}$ on n variables, which is interval with respect to some order of variables $x_1, \dots, x_n$. Let us suppose, that $\mathcal{F}$ represents an interval $[a, b] \subseteq [0, 2^n - 1]$. Clearly both DNFs $\mathcal{F}[x_1 := 1]$ and $\mathcal{F}[x_1 := 0]$ are interval with respect to the order of variables $x_2, \dots, x_n$. If $\mathcal{F}[x_1 := 1]$ is empty, it only means, that $\mathcal{F}$ represents an interval, which is contained in $[0, 2^{n-1} - 1]$, in particular it means that $a, b < 2^{n-1}$, the precise values of a, b can be found by inspecting DNF $\mathcal{F}[x_1 := 0]$. If, on the other hand, $\mathcal{F}[x_1 := 0]$ is empty, then the whole interval represented by $\mathcal{F}$ is contained in $[2^{n-1}, 2^n)$ and in particular $a, b \geq 2^{n-1}$, again the precise values of a, b can be found by inspecting DNF $\mathcal{F}[x_1 := 1]$. Now let us suppose, that neither $\mathcal{F}[x_1 := 0]$ nor $\mathcal{F}[x_1 := 1]$ is empty. In this case $a < 2^{n-1} \leq b$ and clearly $\mathcal{F}[x_1 := 0]$ represents a positive interval function with respect to order $x_2, \dots, x_n$ and $\mathcal{F}[x_1 := 1]$ is a negative interval function with respect to the same order $x_2, \dots, x_n$.

Since we know neither the order $x_1, \dots, x_n$ nor the fact whether $\mathcal{F}$ is interval, we have to be a bit more careful and try to uncover the order step by step. The *while* loop (steps 3 – 17) seeks at each of its tests a variable y, which would make either $\mathcal{F}[y := 1]$ or $\mathcal{F}[y := 0]$ empty. This variable can serve us as the next one in the order. If no such variable exists, then we know, that if we choose any variable y as the next one in the order being constructed and if we have chosen correctly, then $\mathcal{F}[y := 0]$ must be a positive interval DNF and $\mathcal{F}[y := 1]$ must be a negative interval DNF. Since we do not know, which variable should be chosen as the next one, we simply try all the remaining variables one by one. We only have to ensure, that both $\mathcal{F}[y := 0], \mathcal{F}[y := 1]$ are interval with respect to the same order of variables. To test this we simply run two instances of Algorithm 4.2 and check, whether we can choose the same variable in both of these instances at each step.

The proof of the correctness of Algorithm 5.3 is split into two lemmas, Lemma 5.4 shows, that the *while* loop in steps 3 – 17 is correct. The correctness of *if* condition in step 18 is shown in Lemma 5.5

Lemma 5.4 *Let f be a Boolean function on n variables and let $\mathcal{F}$ be its prime and irredundant DNF representation. Let y be such variable, that $\mathcal{F}[y := 0] = \emptyset$ ($\mathcal{F}[y := 1] = \emptyset$ resp.). Then f is interval if and only if $\mathcal{F}[y := 1]$ ($\mathcal{F}[y := 0]$ resp.) represents an interval function. Moreover if $\mathcal{F}[y := 1]$ ($\mathcal{F}[y := 0]$ resp.) is interval with respect to some order of variables $x_2, \dots, x_n$, then $\mathcal{F}$ is interval with respect to $y, x_2, \dots, x_n$.*

Proof : Let us suppose without loss of generality, that $\mathcal{F}[y := 0] = \emptyset$, the case when $\mathcal{F}[y := 1] = \emptyset$ is then similar.

(*if part*) Let us at first assume that $\mathcal{F}[y := 1]$ is an interval function with respect to some order $x_2, \dots, x_n$, where $x_2, \dots, x_n$ are all variables different from y. We claim, that $\mathcal{F}$

is interval with respect to order $y = x_1, x_2, \ldots, x_n$. Indeed, if it is not the case, then there are some three vectors $\vec{u}, \vec{v}, \vec{w} \in \{0,1\}^n$, such that $\vec{u} < \vec{v} < \vec{w}$ lexicographically with respect to order $x_1, \ldots, x_n$, and

$$\begin{aligned}
\mathcal{F}[x_1 := u_1, x_2 := u_2, \ldots, x_n := u_n] &= 1, \\
\mathcal{F}[x_1 := v_1, x_2 := v_2, \ldots, x_n := v_n] &= 0, \\
\mathcal{F}[x_1 := w_1, x_2 := w_2, \ldots, x_n := w_n] &= 1.
\end{aligned}$$

Thus $u_1 = 1$ and $w_1 = 1$ and hence also $v_1 = 1$, which means, that

$$\begin{aligned}
\mathcal{F}[x_1 := 1][x_2 := u_2, \ldots, x_n := u_n] &= 1, \\
\mathcal{F}[x_1 := 1][x_2 := v_2, \ldots, x_n := v_n] &= 0, \\
\mathcal{F}[x_1 := 1][x_2 := w_2, \ldots, x_n := w_n] &= 1.
\end{aligned}$$

This is a contradiction to the fact that $\mathcal{F}[x_1 := 1]$ is an interval function with respect to the order $x_2, \ldots, x_n$.

(*only if part*) Now let us suppose that f is an interval function. According to Lemma 5.2, $\mathcal{F}[y := 1]$ represents an interval function. The same arguments as in the *if part* show, that also in this case f is an interval function to some order $y, x_2, \ldots, x_n$, where $\mathcal{F}[y := 1]$ is an interval function with respect to the order $x_2, \ldots, x_n$. ■

Lemma 5.5 *Let f be a Boolean function on n variables and let $\mathcal{F}$ be a DNF, which represents f. If $\mathcal{F}[y := 0] \neq \emptyset$ and $\mathcal{F}[y := 1] \neq \emptyset$ holds for each variable y, then f is interval if and only if there is a variable y such that $\mathcal{F}[y := 0] \in \mathcal{C}^+_{int}$, $\mathcal{F}[y := 1] \in \mathcal{C}^-_{int}$ and both these DNFs are interval with respect to the same order of variables $x_1, \ldots, x_n$.*

Proof : (*only if part*) Let us at first assume that f is interval with respect to some order $x_1, \ldots, x_n$. The fact that $\mathcal{F}[x_1 := 0] \neq \emptyset$ implies $f[x_1 := 0] \not\equiv 0$. Similarly $f[x_1 := 1] \not\equiv 0$. Since f is an interval function, it must be the case that $f[x_1 := 0, x_2 := 1, \ldots, x_n := 1] = 1$ and similarly $f[x_1 := 1, x_2 := 0, \ldots, x_n := 0] = 1$. Hence $f[x_1 := 0]$ must be a positive interval function with respect to the order $x_2, \ldots, x_n$ and similarly $f[x_1 := 1]$ must be a negative interval function with respect to the same order $x_2, \ldots, x_n$. Clearly x_1 is the variable we are looking for.

(*if part*) Now let us suppose, that there is a variable y and an order $x_2, \ldots, x_n$ of the remaining variables such that $\mathcal{F}[y := 0]$ is a positive interval function with respect to order $x_2, \ldots, x_n$ representing an interval $[a', 2^{n-1} - 1]$, and that $\mathcal{F}[y := 1]$ is a negative interval function with respect to order $x_2, \ldots, x_n$ representing an interval $[0, b']$. Then clearly $\mathcal{F}$ represents the interval $[a = 0a', b = 1b']$ with respect to the order $x_1 = y, x_2, \ldots, x_n$. ■

Theorem 5.6 *Algorithm 5.3 works correctly and terminates in time $O(l \cdot n)$, where l is the length (number of literals) of the input formula.*

Proof : The correctness of the algorithm follows using Lemma 5.4, Lemma 5.5, and the induction on the number of steps of Algorithm 5.3.

We shall proceed by induction on i and show, that before we proceed with step 18, $\mathcal{F}_i$ is an interval DNF with respect to some order $x_i, \ldots, x_n$ if and only if $\mathcal{F}$ is interval with respect to order $x_1, \ldots, x_n$. The proposition clearly holds for $i = 1$, since $\mathcal{F}_1 = \mathcal{F}$. Let us suppose, that the proposition holds for $1, \ldots, i$ and let us show, that it is also true for $i + 1$. By the induction hypothesis, $\mathcal{F}_i$ is an interval DNF with respect to some order $x_i, \ldots, x_n$ if and only if $\mathcal{F}$ is interval with respect to order $x_1, \ldots, x_n$. Now we have to distinguish two cases depending on whether the next variable is picked on line 3 or line 18. In the former case the variable y chosen as x_i in step 3 satisfies conditions of Lemma 5.4, and hence $\mathcal{F}_i$ is an interval DNF with respect to some order $y = x_i, \ldots, x_n$ if and only if $\mathcal{F}_{i+1} = \mathcal{F}[y = x_i := a_i]$ is interval with respect to $x_{i+1}, \ldots, x_n$. The *if* part follows from Lemma 5.4, the *only if* part from Lemma 5.2 It follows, that after the *while* cycle (steps 3 – 17). $\mathcal{F}_i$ is interval if and only if $\mathcal{F}$ is interval. In the latter case, if the next variable is picked in step 18 the statement of this theorem follows directly from Lemma 5.5.

The time requirements of the algorithm are quite straightforward, test in step 3 can be implemented in such a way, that it takes only $O(n)$ time, since e.g. test whether $\mathcal{F}[y := 0]$ only means that we ask whether y is present as a positive literal in every term of $\mathcal{F}$, if we have an array in which we store for each variable the number of terms, in which it appears positively and another array counting for each variable the number of terms, in which it appears negatively, then the appropriate y can be found in $O(n)$ time. These arrays can be updated during partial assignment without changing its asymptotic time requirement $O(l)$. Hence the *while* cycle (steps 3 – 17) requires together time $O(n \cdot l)$.

It only remains to describe, how to perform the *if* test in step 18. For each y we run two instances of Algorithm 4.2 in parallel for $\mathcal{F}[y := 0]$ and $\mathcal{F}[y := 1]$ and if we are choosing the next variable in the order, we ensure, that this variable can be chosen in both instances. It should be clear from the fact that Algorithm 4.2 is correct (Theorem 4.3) and the way how it works, that this approach is correct as well. In order to achieve desired time requirements, we can proceed as follows. Let $\mathcal{F}_0$ denote $\mathcal{F}[y := 0]$, similarly let $\mathcal{F}_1$ denote $\mathcal{F}[y := 1]$. Let I_0 denote the set of variables, which appear in every term of $\mathcal{F}[y := 0]$ or which appear as a linear term in $\mathcal{F}[y := 0]$. Let I_1 denote similar set for $\mathcal{F}[y := 1]$. Note, that the sets I_0 and I_1 together with DNFs $\mathcal{F}_0$ and $\mathcal{F}_1$ may change in every step of Algorithm 4.2. Let us denote $I = I_0 \cap I_1$, this set then consists of exactly those variables, which can be chosen as the next ones in the order of variables with respect to which $\mathcal{F}$ is interval. Now we shall describe, how to maintain the set I throughout the parallel running of two instances of Algorithm 4.2. At each step when we choose a variable from I and perform appropriate partial assignments in $\mathcal{F}_0$ and $\mathcal{F}_1$, some new variables may be added into I_0 and I_1. Note, that the instance of Algorithm 4.2 for $\mathcal{F}_0$ implemented as described in the proof of Theorem 4.4 already implicitly maintains the set I_0, since this set consists of appropriate prefix of the list $Vars$ and of the appropriate prefix of the list $Terms$, hence it suffices to remember indices of the first nonlinear terms in $Terms$ and the first variable which does not appear in all terms of $\mathcal{F}_0$ in $Vars$. If we perform a partial assignment in $\mathcal{F}_0$, these indices are incremented so that they point to the appropriate elements. For each variable newly added to I_0 we test, whether it is also present in I_1, which can be tested using the data structures of $Vars$ and $Terms$ in constant time. The same can be told about $\mathcal{F}_1$ and I_1. Hence for each such variable we can test, whether it should be added into I in constant time. It should be clear that maintenance of I can be implemented in such a way that it consists of two $O(1)$ tests per variable, once it is added into I_0, once it is added into I_1, since the variable is removed from I_1 or I_0 only in the case, it is used as the next one in the order being constructed. Summing over all the variables, the maintenance of I takes $O(n)$ over the entire runs of two instances of Algorithm 4.2. According to Theorem 4.4, the runs of these two parallel instances itself take $O(l)$ time, if appropriately implemented. The fact that we are not generally using the first elements of the lists $Vars$ and $Terms$ does not imply any additional time requirements because $Vars$ and $Terms$ are implemented as double linked lists. Since we run them for each possible y in the worst case, the test in step 18 takes $O(n \cdot l)$ time.

Since each of the remaining steps takes linear time, the time requirements follow. ■

Proof : (*of Theorem 5.1*) Since $\mathcal{C}$ is a class for which we can decide the satisfiability problem in polynomial time, and which is closed under partial assignment, we are able to check for every term t, whether t is an implicant of $\mathcal{F}$. Hence we can transform $\mathcal{F}$ into a prime and irredundant DNF $\mathcal{F}'$ representing f in polynomial time by deleting redundant terms and "shortening" the remaining terms into prime implicants (we refer the reader to [5], where this easy fact was shown for the special case of Horn functions). Then we can test whether $\mathcal{F}'$ represents an interval function using Algorithm 5.3. ■

6 Decomposition of Interval Boolean Functions

In this section we shall show, that each interval function f can be described as a conjunction of a positive interval function g_P and a negative interval function g_N, and that given a prime and irredundant DNF $\mathcal{F}$ representing f, we can easily find a positive DNF $\mathcal{F}^P$ and a negative DNF $\mathcal{F}^N$, such that $\mathcal{F} = \mathcal{F}^P \wedge \mathcal{F}^N$. We shall start with a very simple observation based on the easy fact, that given an interval function f which represents interval $[a, b]$, f can be expressed as a conjunction of a positive interval function which represents interval $[a, 2^n - 1]$ and a negative interval function which represents interval $[0, b]$.

Observation 6.1 *Each interval Boolean function f can be written as $f = g_N \wedge g_P$ where g_N is a negative interval function and g_P is a positive interval function.*

Given a term t we shall denote by t^N a term consisting of all negative literals of t, similarly we shall denote by t^P a term consisting of all positive literals of t. Given a DNF $\mathcal{F} = \bigvee_{i=1}^m t_i$, we shall denote $\mathcal{F}^N = \bigvee_{i=1}^m t_i^N$ and $\mathcal{F}^P = \bigvee_{i=1}^m t_i^P$.

Lemma 6.2 *If $f = g_N \wedge g_P$ where g_N is a negative function and g_P is a positive function, and $\mathcal{F}$ is a DNF representing f, then $\mathcal{F}' = \mathcal{F}^N \wedge \mathcal{F}^P$ is a representation of f. Moreover, if $\mathcal{F}$ is prime then $\mathcal{F}^N$ and $\mathcal{F}^P$ are also both prime DNFs.*

Proof : Let us at first show that $\mathcal{F}'$ represents f. To see that $\mathcal{F} \leq \mathcal{F}'$ it suffices to observe that each term $t \in \mathcal{F}$ is directly present in $\mathcal{F}'$ since $t^N \in \mathcal{F}^N$, $t^P \in \mathcal{F}^P$, and $t = t^N \wedge t^P$, hence t is an implicant of $\mathcal{F}'$.

Now let us show the opposite inequality, i.e. that $\mathcal{F} \geq \mathcal{F}'$. To see this we shall observe that $\mathcal{F}^N \leq g_N$ and $\mathcal{F}^P \leq g_P$. Given a $t^N \in \mathcal{F}^N$, we know that there is a term $t = t^N \wedge t^P \in \mathcal{F}$. Let $\mathcal{F}_N$ be the only prime representation of g_N, and let $\mathcal{F}_P$ be the only prime representation of g_P. Clearly $\mathcal{F} = \mathcal{F}_N \wedge \mathcal{F}_P$ is (after multiplying out $\mathcal{F}_N$ and $\mathcal{F}_P$) a DNF representation of f. Since t is an implicant of f, there is some term $t' \geq t$ which is a prime implicant of f, hence t' can be generated by a series of consensuses from $\mathcal{F}$. We shall proceed by induction on the length of consensus derivation of t' from $\mathcal{F}$ to show, that t'^N is an implicant of $\mathcal{F}_N$ and t'^P is an implicant of $\mathcal{F}_P$. If t' is directly present in $\mathcal{F}$, then the proposition trivially follows from the definition of $\mathcal{F}$. Now let $t_1 = t_1^N \wedge \overline{a} \wedge t_1^P$ and $t_2 = t_2^N \wedge t_2^P \wedge a$ be two terms with consensus, such that $t' = cons(t_1, t_2) = t_1^N \wedge t_2^N \wedge t_1^P \wedge t_2^P$. By the induction hypothesis, we may assume that $t_1^N \wedge \overline{a}$, t_2^N are both implicants of $\mathcal{F}_N$ and t_1^P, $t_2^P \wedge a$ are both implicants of $\mathcal{F}_P$. Clearly $t'^N = t_1^N \wedge t_2^N \leq t_2^N$ and hence t'^N is an implicant of $\mathcal{F}_N$. Similarly $t'^P = t_1^P \wedge t_2^P \leq t_1^P$ and hence t'^P is an implicant of $\mathcal{F}_P$. Since t'^N is an implicant of $\mathcal{F}_N$ and $t^N \leq t'^N$, also t^N is an implicant of $\mathcal{F}_N$, clearly in this case $\mathcal{F}^N \leq \mathcal{F}_N$, similarly $\mathcal{F}^P \leq \mathcal{F}_P$, hence $\mathcal{F}' = \mathcal{F}^N \wedge \mathcal{F}^P \leq g_N \wedge g_P = f$.

Now consider that $\mathcal{F}$ is prime and let us show that in this case $\mathcal{F}^N$ and $\mathcal{F}^P$ are both prime DNFs. Let us assume by a contradiction that there is some $t^N \in \mathcal{F}^N$ which is not prime, hence there is a t'^N which is also an implicant of $\mathcal{F}^N \leq g_N$ and $t'^N > t^N$. Since $\mathcal{F}^N$ is a negative DNF, also $t'^N \in \mathcal{F}^N$. Since $t^N \in \mathcal{F}^N$, there is some $t^P \in \mathcal{F}^P$ for which $t^N \wedge t^P \in \mathcal{F}$. Since $\mathcal{F}^N \wedge \mathcal{F}^P$ is a representation of f, $t'^N \wedge t^P$ is an implicant of f, too. This is clearly a contradiction to the primality of $\mathcal{F}$. ∎

Note that since $\mathcal{F}^N$ is a prime negative DNF, g_N is a negative function, and $\mathcal{F}^N \leq g_N$, $\mathcal{F}^N$ is in fact a sub-DNF of the only prime DNF representation of g_N. The same fact can be observed about g_P.

7 Renamable Interval Functions

It is not hard to see, that interval functions are not closed under variable complementation. Consider e.g. a function defined by the following DNF:

$$\mathcal{F} = b\overline{a} \vee c\overline{a} \vee a\overline{b} \vee c\overline{b} = (a \vee b \vee c)(\overline{a} \vee \overline{b})$$

This DNF clearly defines an interval function with respect to the alphabetical order of variables which represents interval $[001, 101]$. However if we complement variable b, we get

$$\mathcal{F}' = \overline{b}\overline{a} \vee c\overline{a} \vee ab \vee cb$$

which does not define an interval function with respect to any order of variables. This can be observed e.g. using Observation 6.1 and Lemma 6.2, since

$$\mathcal{F}' \neq (\overline{b}\overline{a} \vee \overline{a})(c \vee ab \vee cb) = \overline{a}(c \vee ab)$$

Hence it is natural to ask, if it is possible to recognize whether a given prime and irredundant DNF can be switched to represent an interval function by changing the polarity of some of its variables. First, we shall formalize the notion of renaming.

Definition 7.1 Let $\mathcal{F}$ be a DNF on n variables and let $S \subseteq \{1, \ldots, n\}$ be an index set indexing a subset of variables. Let us define $\mathcal{F}^S$ to be a DNF which is produced from $\mathcal{F}$ by replacing all occurences of x_i by $\overline{x}_i$ and all occurences of $\overline{x}_i$ by x_i for every $i \in S$, and by leaving all other literals (corresponding to variables x_i, $i \notin S$) unchanged. We say, that a DNF $\mathcal{F}$ is *renamable (positive) interval DNF* if there is some index set S, such that $\mathcal{F}^S$ is a (positive) interval DNF. A function f is called *renamable (positive) interval function* if there is some renamable (positive) interval DNF $\mathcal{F}$ which represents f. We shall denote by $\mathcal{C}^R_{int}$ the class of renamable interval functions, by $\mathcal{C}^{R+}_{int}$ the class of renamable positive interval functions, and by $\mathcal{C}^{R-}_{int}$ the class of renamable negative interval functions.

Let us at first observe, that recognition of renamable positive (or negative) interval DNFs is easy.

Lemma 7.2 *Let $\mathcal{F}$ be a prime DNF on n variables and let S be an index set which consists of indices of all variables which do appear negatively in $\mathcal{F}$. Then $\mathcal{F}$ is a renamable positive interval DNF if and only if $\mathcal{F}^S$ is a positive interval DNF.*

Proof : The proof rests on the fact that a prime DNF of a positive function contains only positive literals. Thus a prime DNF $\mathcal{F}$ is renamable positive if and only if every variable has the property, that all its occurences in $\mathcal{F}$ are positive literals or all its occurences in $\mathcal{F}$ are negative literals. Variables that fulfil the latter constitute the set S. ■

Proposition of Lemma 7.2 can be easily reformulated for negative functions.

Corollary 7.3 *Let $\mathcal{F}$ be a prime DNF on n variables and let S be an index set which consists of indices of all variables which do appear positively in $\mathcal{F}$. Then $\mathcal{F}$ is a renamable negative interval DNF if and only if $\mathcal{F}^S$ is a negative interval DNF.*

Moreover, the properties of positive and negative prime DNF yield the folowing statement.

Lemma 7.4 *Let $\mathcal{F}$ be a DNF on n variables, then $\mathcal{F} \in \mathcal{C}^{R+}_{int}$, if and only if $\mathcal{F} \in \mathcal{C}^{R-}_{int}$*

Proof : Clearly, $\mathcal{F}^S$ is a positive interval function (implying $\mathcal{F} \in \mathcal{C}^{R+}_{int}$) if and only if $\mathcal{F}^T$, where $T = \{1, ..., n\} \setminus S$, is a negative interval function (implying $\mathcal{F} \in \mathcal{C}^{R-}_{int}$). ■

Now we shall show, that the recognition of general renamable interval DNFs can be done using only a slight modification of Algorithm 5.3. Note, that the *while* cycle in steps 3 – 17 need not to be changed since it does not matter, whether $\mathcal{F}_i[y := 0] = \emptyset$ or $\mathcal{F}_i[y := 1] = \emptyset$, these cases are completely symmetrical. Step 18 will now test, whether $(\exists y)$ $(\mathcal{F}_i[y := 0] \in \mathcal{C}^{R+}_{int} \wedge \mathcal{F}_i[y := 1] \in \mathcal{C}^{R-}_{int}$ and both are renamable interval functions with respect to the same renaming and order of variables. Due to Lemma 7.4 it is not necessary to test the case when $(\mathcal{F}_i[y := 1] \in \mathcal{C}_{int}^{R+} \wedge \mathcal{F}_i[y := 0] \in \mathcal{C}_{int}^{R-}$. Thus, with the above described small modification of step 18, Algorithm 5.3 tests for a given DNF $\mathcal{F}$ whether $\mathcal{F} \in \mathcal{C}_{int}^{R}$. Hence we have shown the following.

Theorem 7.5 *Given a prime and irrerundant DNF $\mathcal{F}$ on n variables, it is possible to check in time $O(n \cdot l)$, whether $\mathcal{F} \in \mathcal{C}_{int}^{R}$.*

Corollary 7.6 *Given a DNF $\mathcal{F}$ on n variables, which belongs to the class for which we can test the satisfiability problem in polynomial time and which is closed under partial assignment, it is possible to check in polynomial time, whether $\mathcal{F} \in \mathcal{C}_{int}^{R}$.*

8 Interval Extensions of pdBfs

Let T and F be two sets of n-bit Boolean vectors such that $T \cap F = \emptyset$. Then we call the pair (T, F) a *partially defined Boolean function (pdBf)*. A Boolean function f defined on n variables *extends* a pdBf (T, F) if it is true on all vectors from T and false on all vectors from F.

Given a class $\mathcal{C}$ of Boolean functions, an *extension problem* for $\mathcal{C}$ can be stated as follows: given an arbitrary pdBf (T, F), does there exist a function $f \in \mathcal{C}$ such that f extends (T, F)? The following algorithm solves the extension problem for the class of positive interval functions.

Algorithm 8.1 EXTENSION($\mathcal{C}_{int}^{+}$)

Input: *A pdBf (T, F)*
Output: *Order $x_{\pi(1)}, \dots, x_{\pi(n)}$ of variables, and n-bit number a, if there is a positive interval function which extends (T, F) and represents an interval $[a, 2^n)$ with respect to the order $x_{\pi(1)}, \dots, x_{\pi(n)}$.* **NO** *otherwise.*

```
 1: $I := \{1, \dots, n\}$
 2: for $i := 1$ to $n$
 3: do
 4:    if $(\exists j \in I)$ $(\forall u \in T)$ $(u_j = 1)$
 5:    then
 6:       $\pi(i) := j$
 7:       $a_i := 1$
 8:       $F := F \setminus \{v \in F \mid v_j = 0\}$
 9:       $I := I \setminus \{j\}$
10:    else if $(\exists j \in I)$ $(\forall v \in F)$ $(v_j = 0)$
11:    then
12:       $\pi(i) := j$
13:       $a_i := 0$
```

```
14:         $T := T \setminus \{u \in T \mid u_j = 1\}$
15:         $I := I \setminus \{j\}$
16:     else
17:         return NO
18:     endif
19: done
```

Algorithm 8.1 is based on similar ideas as Algorithm 4.2. The index set I contains at each time indices, which have not been considered yet and we restrict our attention only to these indices. Algorithm 8.1 looks at every step for an index j, which would satisfy conditions in steps 4 or 10. If the condition in step 4 is satisfied, it means that each truepoint has the j-th bit equal to 1. In this case we assume, that a positive interval extension f which is being constructed satisfies that if $f(x) = 1$ for some vector x, then $x_j = 1$, and hence conversely if $x_j = 0$ then $f(x) = 0$. Due to this assumption we can discard all the falsepoints v for which $v_j = 0$, which we do in step 8. The similar situation is, when each falsepoint v has $v_j = 0$, in this case we assume, that the extension has the similar property, in particular, that if $f(x) = 0$ for some vector x, then $x_j = 0$, and hence if $x_j = 1$, then $f(x) = 1$. Due to this assumption we can discard all the truepoints u for which $u_j = 1$ (step 14), since these are not interesting any more.

The proof of the correctness of Algorithm 8.1 rests on lemmas 8.2, 8.3, and 8.4. The first one shows that the actions which follow a successful test in step 4 are correct, the second one shows that the actions which follow a successful test in step 10 are correct, and finally the third one proves that if neither of the two tests succeeds then there exists no positive interval extension of the given input.

Lemma 8.2 *Let (T, F) be a pdBf and let us assume that $u_1 = 1$ holds for all $u \in T$. Let T' consist of all vectors from T with the first bit omitted, let F' consist of all vectors v from F with $v_1 = 1$ and with the first bit omitted. Then (T, F) has a positive interval extension if and only if (T', F') has a positive interval extension. Moreover if (T', F') has a positive interval extension f' with respect to some order of variables $x_2, \dots, x_n$, then function $f(x) = x_1 \wedge f'(x')$ (where x' is the vector x with the first bit omitted) is a positive interval extension of (T, F) with respect to the order of variables $x_1, x_2, \dots, x_n$.*

Proof : (*only if* part) Let us at first assume, that (T, F) has a positive interval extension f, the function $f' = f[x_1 := 1]$ clearly extends (T', F'), moreover f' is an interval function according to Lemma 5.2.

(*if* part) Now, let us suppose, that (T', F') has an extension f' which is a positive interval function with respect to some order $x_2, \dots, x_n$. We shall show, that $f(x) = x_1 \wedge f'(x')$ is a positive interval extension of (T, F). To see, that f is a positive interval function it suffices to take a positive interval DNF $\mathcal{F}'$ representing f', now DNF $\mathcal{F} = x_1 \wedge \mathcal{F}'(x')$ is a DNF representing f, such that x_1 is a variable which appears in every term of $\mathcal{F}'$. $\mathcal{F}$ is a positive interval DNF because Algorithm 4.2 would recognize it so, and according to Theorem 4.3, Algorithm 4.2 is correct. Hence it remains to show, that f is an extension of (T, F), which is quite clear. Given a vector $u \in F$, either $u_1 = 0$, in which case $f(u) = 0$, or $u_1 = 1$, in which case $u' \in F'$, $f(u) = f'(u')$, and $f'(u')$ extends (T', F'), hence also in this case $f(u) = 0$. Given $v \in T$, surely $v_1 = 1$ holds, and thus $f(v) = f'(v')$. Since f' extends (T', F') and $v' \in T'$, we have $f(v) = f'(v') = 1$, which completes the proof. ∎

Lemma 8.3 *Let (T, F) be a pdBf and let us assume that $v_1 = 0$ holds for all $v \in F$. Let F' consist of all vectors from F with the first bit omitted, let T' consist of all vectors u from T with $u_1 = 0$ and with the first bit omitted. Then (T, F) has a positive interval extension if and only if (T', F') has a positive interval extension. Moreover if (T', F') has a positive interval extension f' with respect to some order of variables $x_2, \ldots, x_n$, then function $f(x) = x_1 \vee f'(x')$ (where x' is the vector x with the first bit omitted) is a positive interval extension of (T, F) with respect to the order of variables $x_1, x_2, \ldots, x_n$.*

Proof : Since the statement and also its proof are "mirror images" of Lemma 8.2 and of its proof, we leave the details to the reader. ∎

Lemma 8.4 *Let (T, F) be a pdBf and let us assume that there exists no index j such that $u_j = 1$ holds for all $u \in T$ or $v_j = 0$ holds for all $v \in F$. Then there exists no positive interval extension of (T, F).*

Proof : The assumption of the lemma implies that no matter which bit j is selected to be the most significant one, there exist two vectors (numbers) x and y starting with value zero in bit j (i.e. $x_j = y_j = 0$) such that $x \in T$ and $y \in F$, and similarly there exist two vectors (numbers) u and v starting with value one in bit j (i.e. $u_j = v_j = 1$) such that $u \in T$ and $v \in F$. However, these four vectors prevent the function from being a positive interval function with respect to any order of variables in which bit j is the most significant one. ∎

Theorem 8.5 *Algorithm 8.1 works correctly and can be implemented so that it requires $O(n \cdot (n + |T| + |F|))$ time.*

Proof : The correctness of the algorithm follows using a simple induction on i. Lemma 8.2, Lemma 8.3, and Lemma 8.4 show, that each step is correct. Note, that each time the algorithm chooses an index j as the next element in the order, variable x_j plays the role of x_1 in lemmas 8.2 and 8.3. Set I contains all the time those indices, which we still take into account, the remaining ones are omitted.

The time requirements depend on the implementation, we shall describe, how to achieve running time $O(n \cdot (|T| + |F|))$. At the begining of the algorithm we shall construct the following data structures, which are similar to the ones used in the linear time implementation of Algorithm 4.2.

- Array A of length n, each element $A[i]$ will contain a pointer to a double linked list of structures representing true points $u \in T$ for which $u_i = 1$. The head of this list will contain the number of its elements. Each element of the list will contain a pointer to the head.

- Array B of length n, each element $B[i]$ will contain a pointer to a double linked list of structures representing false points $v \in F$ for which $v_i = 0$. the head of this list will contain the number of its elements. Each element of the list will contain a pointer to the head.
- Each vector $u \in T \cup F$ will be represented by a structure S_u. This structure will contain an array P_u of length n. If $u \in T$ and $u_i = 1$, then $P_u[i]$ points to the element of the list representing u in $A[i]$. If $u \in F$ and $u_i = 0$, then $P_u[i]$ points to the element of the list representing u in $B[i]$. In another case $P_u[i]$ contains a nil value. Moreover each element of list in $A[i]$, $B[i]$ representing a vector u will point to the structure S_u.

It should be clear that these data structures can be set up in $O(n \cdot (|T| + |F|))$ time. Using these data structures, we can test the condition in step 4 in $O(n)$ time, since we simply look at each element in array A and ask whether the number of elements of the appropriate list is $|I|$. Similarly, the condition in step 10 can be tested in $O(n)$ time. Both tests are performed at most n times and hence they together require $O(n^2)$ time.

In step 8 we look at $B[j]$ and delete all structures for vectors which are present in the list in $B[j]$, each structure is deleted from all lists in which it is present. This can be done using the pointer to the S_v structure. A similar observation can be made about step 14. Total time requirements of all deletes from our data structures can be at most as high as the total number of elements in all lists, which is $O(n \cdot (|T| + |F|))$, this time is independent on the *for* cycle.

The remaining steps take constant time, since they are repeated n times, they together take $O(n)$ time. ■

The following algorithm solves the extension problem for the class of interval functions. For a set S of n-bit vectors, index $j = 1, \dots, n$, and the set of indices $I \subseteq \{1, \dots, n\}$ let us denote by $S|_{j=e} = \{u \in S \mid u_j = e\}$, where $e \in \{0, 1\}$, and by $S^I = \{u \text{ restricted to bits from } I \mid u \in S\}$.

Algorithm 8.6 EXTENSION($\mathcal{C}_{int}$)

Input: *A pdBf* (T, F)

Output: *Order* $x_{\pi(1)}, \dots, x_{\pi(n)}$ *of variables, and* n*-bit numbers* a, b*, if there is an interval function which extends* (T, F) *and represents an interval* $[a, b]$ *with respect to the order* $x_{\pi(1)}, \dots, x_{\pi(n)}$. **NO** *otherwise.*

```
1: $i := 1$
2: $I := \{1, \dots, n\}$
3: while $(\exists j \in I)$ $[(\forall u \in T)$ $(u_j = 1) \vee (\forall u \in T)$ $(u_j = 0)]$
4: do
5:    if $(\forall u \in T)$ $(u_j = 1)$
6:    then
7:       $a_i := 1$
```

```
 8:         $b_i := 1$
 9:         $F := F \setminus \{v \in F \mid v_j = 0\}$
10:     else
11:         $a_i := 0$
12:         $b_i := 0$
13:         $F := F \setminus \{v \in F \mid v_j = 1\}$
14:     endif
15:     $\pi(i) := j$
16:     $I := I \setminus \{j\}$
17:     $i := i + 1$
18: done
19: for $j \in I$
20: do
21:     $T_0 := (T|_{j=0})^{I\setminus\{j\}}$
22:     $F_0 := (F|_{j=0})^{I\setminus\{j\}}$
23:     $T_1 := (T|_{j=1})^{I\setminus\{j\}}$
24:     $F_1 := (F|_{j=1})^{I\setminus\{j\}}$
25:     if $(T_0, F_0)$ has a positive interval extension $f_0$ representing interval $[a', 2^{n-i})$ and
        $(T_1, F_1)$ has a negative interval extension $f_1$ representing interval $[0, b']$ and both
        extensions are interval with respect to the same order $x_{i+1}, \dots, x_n$.
26:     then
27:         $\pi(i) := j$
28:         $a_i := 0$
29:         $b_i := 1$
30:         $a := a_1 \dots a_i a'$
31:         $b := b_1 \dots b_i b'$
32:         return order $x_{\pi(1)}, \dots, x_{\pi(n)}$ and the numbers $a, b$.
33:     endif
34: done
35: return NO
```

The proof of the correctness of Algorithm 8.6 is very similar to the proof of the correctness of Algorithm 5.3, since both algorithms share the same ideas which they are based on.

Lemma 8.7 *Let (T, F) be a pdBf on n variables and let us assume that $u_1 = 1$ ($u_1 = 0$ resp.) holds for all $u \in T$. Let T' consist of all vectors from T with the first bit omitted, let F' consist of all vectors v from F with $v_1 = 1$ ($v_1 = 0$ resp.) and with the first bit omitted. Then (T, F) has an interval extension if and only if (T', F') has an interval extension. Moreover if (T', F') has an interval extension f' with respect to some order of variables $x_2, \dots, x_n$, then function $f(x) = x_1 \wedge f'(x_2, \dots, x_n)$ ($f(x) = \overline{x}_1 \wedge f'(x_2, \dots, x_n)$ resp.) is a interval extension of (T, F) with respect to the order of variables $x_1, x_2, \dots, x_n$.*

Proof : We shall only show the case when all vectors u in T have the first bit equal to 1, the latter case is analogous. The proof of this lemma is just a modification of the proof of Lemma 8.2

(*only if part*) Let us at first assume, that (T, F) has an interval extension f, the function $f' = f[x_1 := 1]$ clearly extends (T', F'), moreover f' is an interval function according to Lemma 5.2.

(*if part*) Now, let us suppose, that (T', F') has an extension f' which is an interval function with respect to some order $x_2, \dots, x_n$. We shall show, that $f(x) = x_1 \wedge f'(x')$ is an interval extension of (T, F). Let us consider a prime and irredundant interval DNF $\mathcal{F}'$ representing f'. Then DNF $\mathcal{F} = x_1 \wedge \mathcal{F}'(x')$ is a prime and irredundant DNF representing f, and moreover it is an interval DNF, since (as can be easily observed), Algorithm 5.3 would recognize it so. Hence f is an interval function. Similarly as in the proof of Lemma 8.2 it can be observed that f extends (T, F), as well. ■

Lemma 8.8 *Let (T, F) be a pdBf on n variables and let us assume, that for any index $i = 1, \dots, n$ there are some vectors $u, v \in T$ such that $u_i = 1$ and $v_i = 0$. Let T_0, F_0, T_1, F_1 be defined as in steps 21-24 of Algorithm 8.6. Then (T, F) has an interval extension if and only if there exists an index j (w.l.o.g. let $j = 1$ after renumbering) such that the following conditions hold:*

1. *(T_0, F_0) has a positive interval extension f_0, and*
2. *(T_1, F_1) has a negative interval extension f_1, and*
3. *f_0 and f_1 are both interval with respect to the same order of variables $x_2, \dots, x_n$.*

Moreover, if f_0 is a positive interval extension of (T_0, F_0) and f_1 is a negative interval extension of (T_1, F_1), then $f(x) = [x_1 \wedge f_1(x_2, \dots, x_n)] \vee [\overline{x}_1 \wedge f_0(x_2, \dots, x_n)]$ is an interval extension of (T, F) with respect to order $x_1, x_2, \dots, x_n$.

Proof : (*only if part*) Let us at first assume that (T, F) has an interval extension f with respect to some order $x_1, \dots, x_n$. Set $j = 1$, $f_0 = f[x_1 := 0]$, and $f_1 = f[x_1 := 1]$. By Lemma 5.2 both f_0 and f_1 are interval functions. It should be clear, that both of them are interval with respect to order $x_2, \dots, x_n$. Moreover, it is obvious that $f(x) = [x_1 \wedge f_1(x_2, \dots, x_n)] \vee [\overline{x}_1 \wedge f_0(x_2, \dots, x_n)]$.

(*if part*) Now, let us assume, that there is some j, for which the conditions 1-3 apply. We shall show, that $f(x) = [x_1 \wedge f_1(x_2, \dots, x_n)] \vee [\overline{x}_1 \wedge f_0(x_2, \dots, x_n)]$ is an interval extension of (T, F). Again to see, that f is an interval function, we shall recall Algorithm 5.3. Given prime and irredundant DNFs $\mathcal{F}_0$ representing f_0 and $\mathcal{F}_1$ representing f_1, $\mathcal{F} = (x_1 \wedge \mathcal{F}_1) \vee (\overline{x}_1 \wedge \mathcal{F}_0)$ would be recognized by Algorithm 5.3 as an interval DNF. Since Algorithm 5.3 is correct due to Theorem 5.6, f is an interval function. It remains to show, that f extends (T, F), which is quite easy. Given $u \in T$, either $u_1 = 1$, in which case u restricted to bits $2, \dots, n$ belongs to T_1 and hence $f_1(u_2, \dots, u_n) = 1$ and also $f(u) = 1$. If $u_1 = 0$, similarly $f_0(u_2, \dots, u_n) = 1$ and $f(u)$ is again 1. The case when $u \in F$ is similar and so we leave the details to the reader.
■

Theorem 8.9 *Algorithm 8.6 is correct and works in time $O(n \cdot (n + |T| + |F|))$.*

Proof : The correctnes of Algorithm 8.6 is a corollary of Lemma 8.7, Lemma 8.8, and simple induction on i.

Time requirements are straightforward, the *while* cycle in (steps 3 – 18) repeats at most n times, all the steps can be performed in $O(n \cdot (n + |T| + |F|))$ time. The same holds for the *for* cycle in (steps 19 – 34), where we use Theorem 8.5 to show, that condition in the *if* statement (step 25) can be tested in $O(n \cdot (n + |T| + |F|))$ time. ■

9 Conclusions

We have shown, that given a DNF $\mathcal{F}$, we are able to test, whether it represents an interval function w.r.t. some order of variables, provided $\mathcal{F}$ belongs to a class of Boolean formulae, for which we can test satisfiability in polynomial time, and which is closed under partial assignment. We have also shown several properties of the class of interval functions, e.g. that it is closed under partial assignment although it is not closed under variable complementation. In Section 6 we have shown, how an interval function and its DNF can be decomposed into a conjunction of a positive and a negative functions. In Section 7 we have shown that a variable complementation closure of interval functions, which we call renamable interval functions, is recognizable under the same assumptions, under which we are able to recognize the class of interval functions. In Section 8 we have shown, that given a pdBf (T, F), we can find an interval extension of (T, F) in polynomial time, if it exists.

However many problems concerning interval functions still remain open. The following list contains some of them:

1. BEST-FIT interval extensions of pdBfs.
   This problem is a generalization of the interval extension problem of pdBf we studied in Section 8. We are looking for interval function f which for given pdBf (T, F) minimizes the number of vectors that are not correctly classified, that means that we want to minimize $|(T(f) \cap F) \cup (F(f) \cap T)|$.

2. Interval extensions of pBmds.
   This is another generalization of interval extension problem of pdBf. Here we want to decide whether a given pBmd (partially defined Boolean function with missing data) has interval extension. In pBmd (T, F) some vectors in $T \cup F$ can have in some positions '*' instead of 0 or 1, and this means that the value of this bit is not determined. There are several variants of this problem, see [2] for more information.

3. Renamable interval extensions.
   In this case we search for an extension of pdBf from the class of renamable interval extensions.

4. Properties of functions representing more than one interval.
   We can generalize the notion of interval function to 2-interval and even k-interval functions and study their properties.

References

[1] BOROS, E., IBARAKI, T., AND MAKINO, K. Error-free and best-fit extensions of partially defined boolean functions. *Information and Computation 140* (1998), 254 – 283.

[2] BOROS, E., T.IBARAKI, AND MAKINO, K. Extensions of partially defined boolean functions with missing data. Tech. Rep. 6-96, RUTCOR Research Report RRR, Rutgers University, New Brunswick, NJ, 1996.

[3] GAREY, M., AND JOHNSON, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[4] GENESERETH, M., AND NILSSON, N. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, CA, 1987.

[5] HAMMER, P., AND KOGAN, A. Horn functions and their dnfs. *Information Processing Letters 44* (1992), 23 – 29.

[6] HUANG, C.-Y., AND CHENG, K.-T. Solving constraint satisfiability problem for automatic generation of design verification vectors. *Proceedings of the IEEE International High Level Design Validation and Test Workshop* (1999).

[7] LEWIN, D., FOURNIER, L., LEVINGER, M., ROYTMAN, E., AND SHUREK, G. Constraint satisfaction for test program generation, 1995.

[8] QUINE, W. The problem of simplifying the truth functions. *Amer.Math.Monthly 59* (1952), 521 – 531.

[9] SCHIEBER, B., GEIST, D., AND AYAL, Z. Computing the minimum dnf representation of boolean functions defined by intervals. *Discrete Applied Mathematics 149* (2005), 154 – 173.