A Solution of the P versus NP Problem

Norbert Blum
Institut für Informatik, Universität Bonn
Friedrich-Ebert-Allee 144, D-53113 Bonn, Germany
email: blum@cs.uni-bonn.de

August 14, 2017

Abstract

Berg and Ulfberg [4] and Amano and Maruoka [2] have used CNF-DNF-approximators to prove exponential lower bounds for the monotone network complexity of the clique function and of Andreev's function. We show that these approximators can be used to prove the same lower bound for their non-monotone network complexity. This implies $P \neq NP$.

1 Introduction and Preliminaries

Understanding the power of negations is one of the most challenging problems in complexity theory. With respect to monotone Boolean functions, Razborov [12] was the first who could shown that the gain, if using negations, can be super-polynomial in comparision to monotone Boolean networks. Tardos [16] has improved this to exponential. For the characteristic function of an NP-complete problem like the clique function, it is widely believed that negations cannot help enough to improve the Boolean complexity from exponential to polynomial. Since the computation of an one-tape Turing machine can be simulated by a non-monotone Boolean network of size at most the square of the number of steps [15, Ch. 3.9], a super-polynomial lower bound for the non-monotone network complexity of such a function would imply $P \neq NP$. For the monotone complexity of such a function, exponential lower bounds are known [11, 3, 1, 10, 6, 8, 4, 2, 7]. But until now, no one could prove a non-linear lower bound for the non-monotone complexity of any Boolean function in NP. An obvious attempt to get a super-polynomial lower bound for the non-monotone complexity of the clique function could be the extension of the method which has led to the proof of an exponential lower bound of its monotone complexity. This is the so-called “method of approximation” developed by Razborov [11]. Razborov [13] has shown that his approximation method cannot be used to prove better than quadratic lower bounds for the non-monotone complexity of a Boolean function. But Razborov uses a very strong distance measure in his proof for the inability of the approximation method. As elaborated in [5], one can use the approximation method with a weaker distance measure to prove a super-polynomial lower bound for the non-monotone complexity of a Boolean function.

Our goal is to extend approximators developed for proving a super-polynomial lower bound for the monotone complexity of a monotone Boolean function such that these can be used to prove a super-polynomial lower bound for the non-monotone complexity of the same Boolean function. Note that not every approximator can be suitable for doing this. In [12], Razborov has constructed approximators to prove an $m^{\Omega(\log m)}$ lower bound for the monotone network complexity of the perfect matching function PM_m. This is a monotone Boolean function of m^2 variables which correspond to the edge set of a bipartite $G = (A, B, E)$ where $|A| = |B| = m$. $\mathrm{PM}_m(x) = 1$ iff the corresponding graph G contains a perfect matching. Since a perfect matching in a given bipartite graph can be computed in polynomial time, the non-monotone complexity of PM_m is also polynomial. Therefore, it is impossible to extend Razborov's approximator developed for PM_m to prove a super-polynomial lower bound for the non-monotone complexity of PM_m.

Before describing approximators which seem to be suitable for the extension to non-monotone networks, we shall give some basic definitions. $\mathcal{B}_n := \{f \mid f : \{0,1\}^n \to \{0,1\}\}$ is the set of all n-ary Boolean functions. The *ith variable* is denoted by $x_i : \{0,1\}^n \to \{0,1\}$, $1 \leq i \leq n$. Let $V_n := \{x_i \mid 1 \leq i \leq n\}$. Variables and negated variables are called *literals*. A function $m : \{0,1\}^n \to \{0,1\}$ which is the conjunction of some literals is called a *monomial*. If we delete some literals from a monomial m then we obtain a *submonomial* of m. The empty monomial is the constant function 1. The disjunction of monomials is a formula in *disjunctive normal form* (DNF). The disjunction of some literals is called a *clause*. If we delete some literals from a clause d then we obtain a *subclause* of d. The empty clause is the constant function 0. The conjunction of clauses is a formula in *conjunctive normal form* (CNF). A monomial m is called an *implicant* of the function f if for all $a \in \{0,1\}^n$, $m(a) = 1$ implies $f(a) = 1$. An implicant m is a *prime implicant* of f if no proper submonomial of m is an implicant of f. A clause d is called an *f-clause* if for all $a \in \{0,1\}^n$, $d(a) = 0$ implies

$f(a) = 0$. A *prime clause* d of f is an f-clause where no proper subclause of d is an f-clause. Let $a := (a_1, a_2, \ldots, a_n)$, $b := (b_1, b_2, \ldots, b_n) \in \{0,1\}^n$. We write $a \leq b$ iff $a_i \leq b_i$ for $1 \leq i \leq n$. A function $f \in \mathcal{B}_n$ is *monotone* iff for all $a, b \in \{0,1\}^n$ there hold $a \leq b$ implies $f(a) \leq f(b)$. Obviously, both constant functions are monotone. Let $\Omega_0 := \{\wedge, \vee, \neg\}$ and $\Omega_m := \{\wedge, \vee\}$. For $\Omega \in \{\Omega_0, \Omega_m\}$, an Ω-*network* β is a directed, acyclic graph such that each node has indegree at most two. The nodes g with indegree zero are *input nodes* and are labelled with $\mathrm{op}(g) \in V_n$. The nodes g with indegree larger than zero are the *gates* of β. Each gate g is labelled with an operator $\mathrm{op}(g) \in \Omega$ where the indegree of g is equal the number of operands of $\mathrm{op}(g)$. A node with outdegree zero is an *output node*. For a node g in β let $\mathrm{pred}(g) := \{h \mid h \rightarrow g \text{ is an edge in } \beta\}$ be the set of its direct predecessors. With each node g, we associate a function $\mathrm{res}_\beta(g) : \{0,1\}^n \rightarrow \{0,1\}$ which is defined as follows:

$$\mathrm{res}_\beta(g) := \begin{cases} \mathrm{op}(g) & g \text{ is an input node,} \\ \neg\mathrm{res}_\beta(h_1) & \mathrm{op}(g) = \neg,\ \mathrm{pred}(g) = \{h_1\}, \\ \mathrm{res}_\beta(h_1)\ \mathrm{op}(g)\ \mathrm{res}_\beta(h_2) & \mathrm{op}(g) \in \{\wedge, \vee\},\ \mathrm{pred}(g) = \{h_1, h_2\}. \end{cases}$$

The functions $\mathrm{res}_\beta(g)$ with g is a node in β are computed by β. Let $f \in \mathcal{B}_n$. The minimum number of gates in an Ω_0-network which computes f where negations are not counted is the *non-monotone complexity* $C(f)$ of f. An Ω_m-network is called a *monotone network*. Note that exactly the non-constant monotone Boolean functions can be computed by a monotone network. The minimum number of gates in a monotone network which computes the monotone function f is the *monotone complexity* $C_m(f)$ of f.

Given any Ω_0-network β, we can convert β to an equivalent Ω_0-network β' where all negations occur only at the input nodes. Moreover, the size of β is at most doubled. For doing this, we start at the output nodes and apply De Morgan rules for bringing the negations to the input nodes. Since gates can be simultaneously negated and non-negated, some gates have to be doubled. The resulting network is a so-called *standard network* where only input variables are negated. We consider a negated variable $\neg x_i$ as an input node g with $\mathrm{op}(g) = \neg x_i$. The *standard complexity* $C_{st}(f)$ of a function $f \in \mathcal{B}_n$ is the size of a smallest standard network which computes f. Note that the standard and the non-monotone complexity of a function f differs at most by the factor two. Hence, for proving a super-linear lower bound for the non-monotone complexity of a Boolean function, we can restrict us to the consideration of standard networks.

Which approximators seems to be suitable for the extention to standard networks? CNF-DNF-approximators are introduced implicitly by Haken [6] and explicitly by Berg and Ulfberg [4] and by Amano and Maruoka [2]. To prove a lower bound for the monotone complexity of a monotone Boolean function f, they approximate for each node g in the monotone network β the function $\text{res}_\beta(g)$ by two approximators, a CNF formula and a DNF formula. To describe the effect of the approximators, they use a set of positive test inputs $T_1 \subseteq f^{-1}(1)$ and a set of negative test inputs $T_0 \subseteq f^{-1}(0)$. They assume that the monotone network β under consideration computes the value $f(c)$ for each test input $c \in T_0 \cup T_1$ correctly. An approximator of a gate g introduces an error for a test input c if, at the output node g_0, $f(c)$ is computed correctly before the approximation but incorrectly after the approximation of $\text{res}_\beta(g)$. They show that at any gate of the monotone network, an error for only "few" test inputs is introduced, but the approximators for the output node compute the value incorrectly for "many" test inputs. Since before the definition of any approximator, the network β computes the value of each test input correctly, the network β must contain "many" gates. Let us summarize the properties used in the lower bound proof.

1. For each test input $c \in T_0 \cup T_1$, the monotone network β computes the value $f(c)$ correctly at its output node g_0.

2. The structure of the approximators allows the proof that the approximation of $\text{res}_\beta(g)$ for a gate g introduce an error for only "few" test inputs.

3. The structure of the approximators allows the proof that the approximators for the output node g_0 compute the values of "many" test inputs incorrectly.

Our goal is to treat the negated variables in a standard network which computes a given non-constant monotone Boolean function f at its output node g_0 in such a way that we can use the approximators developed for f with respect to monotone networks on standard networks.

In [14], Razborov and Rudich have introduced the notion of "natural proof". They have shown that natural proofs cannot be used for separating P and NP unless hard pseudorandom generators do not exist. They also mention: "Another exception to our scheme is the list of strong lower bounds proofs against *monotone* circuit models. Here the issue is not constructivity–the properties used in these proofs are all feasible–but that there appears to be no good formal analogue of the largeness condition. In particular, no one has formulated a workable definition of a "random monotone function"." Our method will only apply to monotone Boolean functions.

In the next section, we shall prove some basic properties of monotone and standard networks. In Section 3, we shall specify CNF-DNF-approximators for monotone Boolean functions from a point of view needed for their use on standard networks. Since the knowledge of the lower bound proof for the monotone complexity of the clique function would be useful to understand the approach, we shall describe Berg and Ulfberg's proof [4] in Section 4. Given any standard network β for any non-constant monotone Boolean function $f \in \mathcal{B}_n$, we shall outline the processing of the negated variables in β leading to "reduced" CNF and DNF formulas in Section 5. The use of CNF-DNF-approximators on standard networks is described in Section 6. In Section 7, we apply CNF-DNF-approximators to prove an exponential lower bound for the standard complexity of the clique function and of Andreev's function leading to the proof that P $\neq$ NP. We shall discuss briefly negations in Boolean networks in Section 8.

2 Some Basic Properties of Monotone and Standard Networks

First we shall describe the DNF and CNF formulas constructed by a monotone or by a standard network. Let β be a monotone or a standard network. Consider any node g in β. The function $\text{res}_\beta(g)$ can be written as a DNF formula; i.e., $\text{res}_\beta(g) = \bigvee_{j=1}^{r} m_j$ where each m_j is a monomial. We denote this formula the *DNF representation* $\text{DNF}_\beta(g)$ of $\text{res}_\beta(g)$. Starting at the input nodes, the network β constructs these DNF formulas in the following way:

1. If g is an input node with $\text{op}(g) = x_i$ or $\text{op}(g) = \neg x_i$ then
$$\text{DNF}_\beta(g) := \text{op}(g).$$
2. If g is an $\vee$-gate with $\text{pred}(g) = \{h_1, h_2\}$ then
$$\text{DNF}_\beta(g) := \text{DNF}_\beta(h_1) \vee \text{DNF}_\beta(h_2).$$
3. If g is an $\wedge$-gate with $\text{pred}(g) = \{h_1, h_2\}$, $\text{DNF}_\beta(h_1) = \bigvee_{i=1}^{t_1} m_i$ and $\text{DNF}_\beta(h_2) = \bigvee_{j=1}^{t_2} m'_j$ then
$$\text{DNF}_\beta(g) := \bigvee_{i=1}^{t_1} \bigvee_{j=1}^{t_2} (m_i \wedge m'_j).$$

Each input $a \in \text{res}_\beta(g)^{-1}(1)$ satisfies a monomial m_j of $\text{DNF}_\beta(g)$. Each input $b \in \text{res}_\beta(g)^{-1}(0)$ does not satisfy any monomial in $\text{DNF}_\beta(g)$. Hence, each monomial in $\text{DNF}_\beta(g)$ contains a variable x_i with $b_i = 0$ or a negated variable $\neg x_j$ with $b_j = 1$.

The function $\text{res}_\beta(g)$ can be written as a CNF formula as well; i.e., $\text{res}_\beta(g) = \bigwedge_{j=1}^{s} d_j$ where each d_j is a clause. We denote this formula the *CNF representation* $\text{CNF}_\beta(g)$ of $\text{res}_\beta(g)$. Starting at the input nodes, the network β constructs these CNF formulas in the following way:

1. If g is an input node with $\text{op}(g) = x_i$ or $\text{op}(g) = \neg x_i$ then
$$\text{CNF}_\beta(g) := \text{op}(g).$$

2. If g is an $\wedge$-gate with $\text{pred}(g) = \{h_1, h_2\}$ then
$$\text{CNF}_\beta(g) := \text{CNF}_\beta(h_1) \wedge \text{CNF}_\beta(h_2).$$

3. If g is an $\vee$-gate with $\text{pred}(g) = \{h_1, h_2\}$, $\text{CNF}_\beta(h_1) = \bigwedge_{i=1}^{t_1} d_i$ and $\text{CNF}_\beta(h_2) = \bigwedge_{j=1}^{t_2} d'_j$ then
$$\text{CNF}_\beta(g) := \bigwedge_{i=1}^{t_1} \bigwedge_{j=1}^{t_2} (d_i \vee d'_j).$$

Each input $b \in \text{res}_\beta(g)^{-1}(0)$ falsifies a clause d_j of $\text{CNF}_\beta(g)$. Each input $a \in \text{res}_\beta(g)^{-1}(1)$ does not falsify any clause in $\text{CNF}_\beta(g)$. Hence, each clause in $\text{CNF}_\beta(g)$ contains a variable x_i with $a_i = 1$ or a negated variable $\neg x_j$ with $a_j = 0$.

A monomial or a clause is *trivial* if it contains both literals with respect to at least one variable and *non-trivial* otherwise. Independently from the concrete values of the variables, such a monomial would always have the value zero and such a clause would always have the value one. By construction, $\text{CNF}_\beta(g)$ can contain trivial clauses and $\text{DNF}_\beta(g)$ can contain trivial monomials. We say that the monomials in $\text{DNF}_\beta(g)$ and the clauses in $\text{CNF}_\beta(g)$ are *constructed* at the node g by the network β.

The following theorem characterizes exactly the DNF and the CNF representations of $\text{res}_\beta(g_0)$ with respect to a monotone or a standard network which computes a Boolean function $f \in \mathcal{B}_n$ at its output node g_0.

Theorem 1 *Let β be a monotone or a standard network which computes a Boolean function $f \in \mathcal{B}_n$ at its output node g_0. Then the following hold:*

a) *Besides trivial monomials, DNF$_\beta(g_0)$ contains only implicants of the function f. Furthermore, for each $a \in f^{-1}(1)$, DNF$_\beta(g_0)$ contains an implicant m_a of f such that $m_a(a) = 1$.*

b) *Besides trivial clauses, CNF$_\beta(g_0)$ contains only f-clauses. Furthermore, for each $b \in f^{-1}(0)$, CNF$_\beta(g_0)$ contains an f-clause d_b such that $d_b(b) = 0$.*

Proof: Assume that DNF$_\beta(g_0)$ contains a non-trivial monomial m which is not an implicant of f. Then, by the definition of an implicant of f, there exists $b \in \{0,1\}^n$ such that $m(b) = 1$ but $f(b) = 0$. This contradicts the assumption that β computes f at its output node g_0. Hence, all non-trivial monomials of DNF$_\beta(g_0)$ are implicants of f.

Assume that there is $a \in f^{-1}(1)$ such that $m(a) = 0$ for all implicants m in DNF$_\beta(g_0)$. Then $\text{res}_\beta(g_0)(a) = 0$ but $f(a) = 1$. This contradicts the assumption that β computes f at its output node g_0. Hence, for each $a \in f^{-1}(1)$, DNF$_\beta(g_0)$ contains an implicant m_a of f such that $m_a(a) = 1$.

This proves part a) of the theorem. Analogously, we can prove part b) of the theorem. □

Note that each implicant of a Boolean function f contains a submonomial which is a prime implicant of f. Furthermore, each f-clause contains a subclause which is a prime clause of f.

Every DNF formula can be transformed into an equivalent CNF formula. To see this let $\alpha = \bigvee_{i=1}^{t} m_i$ be a DNF formula which computes a Boolean function $f \in \mathcal{B}_n$. To obtain an equivalent CNF formula γ, we pick from each monomial m_i, $1 \leq i \leq t$ one literal and perform the disjunction of all chosen literals. Then the conjunction of all clauses which can be constructed in this way is a CNF formula $\gamma = \bigwedge_{j=1}^{s} d_j$ which corresponds to the DNF formula α. The following lemma shows that γ computes the function f.

Lemma 1 *Let $\alpha = \bigvee_{i=1}^{t} m_i$ be a DNF formula which computes a Boolean function $f \in \mathcal{B}_n$. Let $\gamma = \bigwedge_{j=1}^{s} d_j$ be the CNF formula constructed from α as described above. Then γ computes f.*

Proof: Consider $a \in f^{-1}(1)$. Then there is a monomial m_l in α such that $m_l(a) = 1$. Since each clause of γ contains a literal of m_l, the input a satisfies all clauses in γ. Hence $\gamma(a) = 1$.

Let $b \in f^{-1}(0)$. Then each monomial in α contains a literal which is not satisfied by b. Consider a clause d_l of γ which picks from each monomial a literal which is not satisfied by b. Obviously, $d_l(b) = 0$. Hence, $\gamma(b) = 0$.

Altogether, we have shown that γ computes f. □

We call such a transformation of a DNF formula to an equivalent CNF formula a *DNF/CNF-switch*. A DNF/CNF-switch can be organized as the construction of a tree T in the following way:

1. Each edge in T is labelled by a literal. With each node w in T we associate the clause $d(w)$ which is obtained by the disjunction of the variables on the unique path from the root of T to w. T is constructed while *expanding* the monomials $m_0, m_1, m_2, \ldots, m_t$ where m_0 is the empty monomial.

2. While expanding m_0, the root T is created. The associated clause is the empty clause.

3. Suppose that w is a leaf that was created while expanding m_i. Then the monomial m_{i+1} is expanded at the leaf w in the following way: The leaf w obtains for each literal in m_{i+1} a new son w'. The edge (w, w') is labelled with the corresponding literal.

After the construction of the tree T, the clauses corresponding to the paths from the root of T to the leaves are the clauses contained in the CNF formula γ obtained from $\alpha = \bigvee_{i=1}^{t} m_i$ by performing a DNF/CNF-switch.

Analogously, every CNF formula can be transformed into an equivalent DNF formula. To see this let $\gamma = \bigwedge_{i=1}^{t} d_i$ be a CNF formula which computes a Boolean function $f \in \mathcal{B}_n$. To obtain an equivalent DNF formula α, we pick from each clause d_i, $1 \leq i \leq t$ one literal and perform the conjunction of all chosen literals. Then the disjunction of all monomials which can be constructed in this way is a DNF formula $\alpha = \bigvee_{j=1}^{s} m_j$ which corresponds to the CNF formula γ. The following lemma shows that α computes the function f.

Lemma 2 *Let $\gamma = \bigwedge_{i=1}^{t} d_i$ be a CNF formula which computes a Boolean function $f \in \mathcal{B}_n$. Let $\alpha = \bigvee_{j=1}^{s} m_j$ be the DNF formula constructed from γ as described above. Then α computes f.*

Proof: Consider $b \in f^{-1}(0)$. Then there is a clause d_l in γ such that $d_l(b) = 0$. Since each monomial of α contains a literal of d_l, the input b falsifies all monomials in α. Hence, $\alpha(b) = 0$.

Let $a \in f^{-1}(1)$. Then each clause in γ contains a literal which is satisfied by a. Consider a monomial m_l of α which picks from each clause a literal which is satisfied by a. Obviously, $m_l(a) = 1$. Hence, $\alpha(a) = 1$.

Altogether, we have shown that α computes f. $\square$

We call such a transformation of a CNF formula to an equivalent DNF formula a *CNF/DNF-switch*. A CNF/DNF-switch can be organized as the construction of a tree analogously to a DNF/CNF-switch.

If m is a monomial in a DNF formula α then we say that α *contains* m. If d is a clause in a CNF formula γ then we say that γ *contains* d. We say that a DNF formula α *contains* a clause d if the CNF formula γ which is obtained by applying a DNF/CNF-switch to α contains d. We say that a CNF formula γ *contains* a monomial m if the DNF formula α which is obtained by applying a CNF/DNF-switch to γ contains m.

Let f be a Boolean function. By the definition of an f-clause and of a prime implicant of f, an input which falsifies an f-clause must also falsify each prime implicant of f. Hence, an f-clause contains at least one literal of each prime implicant of f. By the definition of a prime clause of f, the removal of any literal yields a clause which is not an f-clause. Therefore, for each literal in a prime clause c of f there is a prime implicant p of f such that this literal is the only literal of p contained in c. Similarly, an implicant of f contains at least one literal of each prime clause of f and for each literal in a prime implicant of f there is at least one prime clause c of f such that this literal is the only literal of c contained in p.

Now we shall consider inputs corresponding exactly to a prime implicant or exactly to a prime clause. Let $f \in \mathcal{B}_n$ be a non-constant monotone Boolean function with prime implicants $p_1, p_2, \ldots, p_t$ and prime clauses $c_1, c_2, \ldots, c_s$. For each prime implicant p_l, $a \in \{0,1\}^n$ such that $a_i = 1$ iff x_i is a variable in p_l is called the *p_l-input* of f. PI(f) denotes the set of all p_l-inputs of f. For each prime clause c_l, $b \in \{0,1\}^n$ such that $b_i = 0$ iff x_i is a variable in c_l is called the *c_l-input* of f. PC(f) denotes the set of all c_l-inputs of f. The following theorem shows that for any non-constant monotone Boolean function $f \in \mathcal{B}_n$, a monotone network β which computes at its output node g_0 the values for all inputs in $PI(f)$ and for all inputs in $PC(f)$ correctly has to compute the function f.

Theorem 2 *Let $f \in \mathcal{B}_n$ be any non-constant monotone Boolean function. Let β be any monotone network with output node g_0. If $res_\beta(g_0)(a) = 1$ for all $a \in PI(f)$ and $res_\beta(g_0)(b) = 0$ for all $b \in PC(f)$ then $res_\beta(g_0) = f$.*

Proof: Since β is a monotone network and all prime implicants of f are implicants of $\mathrm{res}_\beta(g_0)$, all implicants of f are also implicants of $\mathrm{res}_\beta(g_0)$. To prove that all implicants of $\mathrm{res}_\beta(g_0)$ are also implicants of f, assume that the monomial m is an implicant of $\mathrm{res}_\beta(g_0)$ but not an implicant of f. Then each prime implicant p_i of f must contain a variable x_j which is not a variable of the monomial m. Otherwise, the monomial m would contain a prime implicant of f as a submonomial and hence, would be an implicant of f. Our goal is to construct a prime clause c_l of f which does not contain any variable of m.

Let $p_1, p_2, \ldots, p_t$ be the prime implicants of f. We consider the prime implicants of f one after another and choose an appropriate variable x_j of p_i to be a variable of the prime clause c_l under construction. Let p_i be the currently considered prime implicant. If no variable of p_i is already chosen to be a variable of c_l then choose any variable of p_i which is not also a variable of m. Obviously, c_l is an f-clause. By construction, no variable can be removed from c_l without destroying this property. Hence, c_l is a prime clause of f. Furthermore, c_l does not contain any variable of m.

Consider the c_l-input $b \in \mathrm{PC}(f)$. Since $b_j = 1$ for all j with x_j is not a variable in c_l, there holds $m(b) = 1$. This contradicts the assumption $\mathrm{res}_\beta(g_0)(b) = 0$. $\square$

Note that a standard network has not this property. To see this consider the characteristic function of the set $\mathrm{PI}(f)$. A standard network β which computes this characteristic function at its output node g_0, computes the values for all inputs in $\mathrm{PI}(f)$ and all inputs in $\mathrm{PC}(f)$ correctly but $\mathrm{res}_\beta(g_0) \neq f$. Hence, we have obtained the following theorem.

Theorem 3 *Let $f \in \mathcal{B}_n$ be any non-constant monotone Boolean function. Then there is a standard network β with output node g_0 such that $res_\beta(g_0)(a) = 1$ for all $a \in PI(f)$ and $res_\beta(g_0)(b) = 0$ for all $b \in PC(f)$ but $res_\beta(g_0) \neq f$.*

Assume that given any monotone or standard network β which computes a Boolean function $f \in \mathcal{B}_n$ at its output node g_0 and an input $c \in \{0,1\}^n$, we wish to evaluate β with input c. For a given input c, there are different methods for the computation of the value computed at the output node g_0. The different methods originate from different points of view.

Method 1:

Starting at the input nodes, the nodes of β are considered in any topological order and evaluated with respect to the input c; i.e., when a gate g is considered then $\mathrm{res}_\beta(h_1)(c)$ and $\mathrm{res}_\beta(h_2)(c)$ of both direct predecessors h_1 and h_2 are known. Hence, $\mathrm{res}_\beta(g)(c)$ can be computed. After the consideration of the output node g_0, the value $f(c) = \mathrm{res}_\beta(g_0)(c)$ is known.

Method 2:

$\mathrm{DNF}_\beta(g_0)$ is constructed first. Then it is checked if $\mathrm{DNF}_\beta(g_0)$ contains an implicant m with $m(c) = 1$. If $\mathrm{DNF}_\beta(g_0)$ contains such an implicant then $f(c) = 1$. Otherwise, $f(c) = 0$.

Method 3:

$\text{CNF}_\beta(g_0)$ is constructed first. Then it is checked if $\text{CNF}_\beta(g_0)$ contains an f-clause d with $d(c) = 0$. If $\text{CNF}_\beta(g_0)$ contains such an f-clause then $f(c) = 0$. Otherwise, $f(c) = 1$.

Obviously, all three methods result in the same value $f(c)$. Haken [6] considers for inputs $a \in f^{-1}(1)$ the flow through those gates g in a monotone network with $\text{res}_\beta(g)(a) = 1$ and for inputs $b \in f^{-1}(0)$ the flow through those gates g with $\text{res}_\beta(g)(b) = 0$. This means that he has used Method 1 for his considerations. Berg and Ulfberg [4] and also Amano and Maruoka [2] have turned Haken's approach into an approximation argument by using CNF-DNF-approximators. For the extension of these approximators such that they can be used on standard networks, it would be more suitable to use the other two methods for the considerations. This means that it would be useful to investigate the effect of the approximations to the construction of certain monomials in the DNF representation of $\text{res}_\beta(g_0)$ and to the construction of certain clauses in the CNF representation of $\text{res}_\beta(g_0)$. Hence, for $a \in f^{-1}(1)$ and $b \in f^{-1}(0)$, we shall investigate subnetworks β_a and β_b of the network β which contructs an implicant m_a with $m_a(a) = 1$ and an f-clause d_b with $d_b(b) = 0$ at the output node g_0 of β. By Theorem 1, for each $a \in f^{-1}(1)$ and each $b \in f^{-1}(0)$, at least one such a subnetwork β_a and at least one such a subnetwork β_b exist. It is possible that β contains more than one such a subnetwork.

To get a subnetwork β_a, we start at the output node g_0 and run backwards through β. For each visited node g, $\text{DNF}_\beta(g)$ contains a monomial $m_a(g)$ which is used for the construction of the monomial m_a at the output node g_0. For the output node g_0, there holds $m_a(g_0) = m_a$. Both direct predecessors h_1 and h_2 of g are contained in β_a iff g is an $\wedge$-gate. If g is an $\vee$-gate then exactly one of its both directed predecessors is contained in β_a. In dependence of the type of g, the predecessors of g in β_a and the corresponding monomials are determined in the following way:

Case 1: g is an $\wedge$-gate.

By construction, $\text{DNF}_\beta(h_1)$ contains a monomial m_1 and $\text{DNF}_\beta(h_2)$ contains a monomial m_2 such that $m_a(g) = m_1 \wedge m_2$. Therefore, $m_a(h_1) = m_1$ and $m_a(h_2) = m_2$.

Case 2: g is an $\vee$-gate.

Note that from exactly one of $\text{DNF}_\beta(h_1)$ and $\text{DNF}_\beta(h_2)$, a monomial m is used for the construction of $m_a(g)$. By construction, there holds $m = m_a(g)$. Then h_i such that a monomial m in $\text{DNF}_\beta(h_i)$ is used for the construction of

$m_a(g)$ is that direct predecessor of g which is contained in β_a. Furthermore, $m_a(h_i) = m = m_a(g)$.

By construction, for each node g in β_a, $m_a(g)$ is defined and there holds $m_a(g)(a) = 1$. For each node g which is not contained in β_a, $m_a(g)$ is undefined. Analogously, we can construct the subnetwork β_b for the input $b \in f^{-1}(0)$.

3 CNF-DNF-Approximators for Monotone Boolean Networks

We shall specify CNF-DNF-approximators as needed for their extension to standard networks. The aim of a CNF-DNF-approximator applied to a given monotone network β is the approximation of the CNF or the DNF representations of the functions $\mathrm{res}_\beta(g)$ computed at the nodes g of the network β. Approximation means that the size of the monomials and the size of the clauses in the DNF and CNF formulas associated with the nodes of β are bounded. The main tools of a CNF-DNF-approximator designed for a given monotone Boolean function are the CNF/DNF- and the DNF/CNF-switches which maintain the size bounds of the monomials and of the clauses contained in the constructed formulas. Although the only intention is the approximation of either the CNF or the DNF formulas associated with the nodes of β, an approximation of the other formulas is needed as well such that CNF/DNF- or DNF/CNF-switches can be applied suitably. Assume that we intend the approximation of the Φ formulas, $\Phi \in \{\mathrm{CNF}, \mathrm{DNF}\}$ associated with the nodes in a monotone Boolean network β which computes a non-constant monotone Boolean function $f \in \mathcal{B}_n$ at its output node g_0.

The idea is to consider the network β in any topological order and to define the approximators corresponding to the considered node g. This means that in the case that g is a gate, the approximators of both direct predecessors h_1 and h_2 are already defined. We approximate the function $\mathrm{res}_\beta(g)$ by two approximators, a DNF formula $\mathcal{D}_g^r$ and a CNF formula $\mathcal{C}_g^k$ where the size of a monomial in $\mathcal{D}_g^r$ is always smaller than r and the size of a clause in $\mathcal{C}_g^k$ is always smaller than k. The size of a monomial or of a clause is its number of distinct variables or another measure. We do not restrict the number of monomials in $\mathcal{D}_g^r$ or the number of clauses in $\mathcal{C}_g^k$. For the construction of $\mathcal{C}_g^k$ and of $\mathcal{D}_g^r$, we use the approximators of the direct predecessors of g.

To describe the effect of the approximators, sets $T_1 \subseteq f^{-1}(1)$ and $T_0 \subseteq f^{-1}(0)$ are used. The elements of T_1 and T_0 are called *positive* and *negative* *test inputs*. Let us consider the moment when a node g is considered for the definition of its approximators. The nodes v of the network β fulfilling the following properties define the *current front* of the network:

1. The approximators with respect to the node v are defined.
2. There is a direct successor w of v such that the approximators for $\text{res}_\beta(w)$ are not defined.

In dependence of the approximators defined for the nodes at the current front, we can compute the DNF and the CNF representations of the current function computed at the output node g_0. Let $\Phi(\beta)$ denote the Φ representation of the current function computed at g_0.

Consider any input $a \in T_1$. We say that the approximators for g *introduce an error for the input* a if before the approximation of $\text{res}_\beta(g)$, $\Phi(\beta)$ contains a monomial m_a which is satisfied by the input a but after the approximation, none such a monomial in $\Phi(\beta)$ exists. Consider any input $b \in T_0$. We say that the approximators for g *introduce an error for the input* b if before the approximation of $\text{res}_\beta(g)$, $\Phi(\beta)$ contains a clause d_b which is falsified by the input b but after the approximation, none such a clause in $\Phi(\beta)$ exists.

CNF-DNF-approximators switch between CNF and DNF formulas. There is an essential difference between the approach of Amano and Maruoka [2] and the approach of Berg and Ulfberg [4]. Amano and Maruoka first perform a CNF/DNF-switch to construct from a CNF formula γ which contains only clauses of size less than k an equivalent DNF formula α and then they delete from the obtained DNF formula α the monomials of size larger than $r-1$. This means that the monomials of larger size than $r-1$ are replaced by zero. The transformation of a DNF formula to a CNF formula is performed analogously. First, a DNF/CNF-switch is performed to construct from a DNF formula α which contains only monomials of size less than r an equivalent CNF formula γ and then they delete from the obtained CNF formula γ the clauses of size larger than $k-1$. This means that the clauses of larger size than $k-1$ are replaced by one.

Berg and Ulfberg have used the fact that it would be sufficient to construct from a CNF formula γ which contains only clauses of size less than k a DNF formula α which computes with respect to all test inputs the same value as γ and then to delete from the obtained DNF formula α the monomials of size larger than $r-1$. More precisely, they use only an appropriate subset of the monomials which would be constructed by the CNF/DNF-switch of γ for the construction of the DNF formula α. The other monomials obtained by the CNF/DNF-switch of γ are replaced by zero. The transformation of a DNF formula to a CNF formula is performed analogously. A DNF formula α and a CNF formula γ are *ti-equivalent* iff for all test inputs $c \in T_1 \cup T_0$, $\alpha(c) = \gamma(c)$. Note that the equivalence of a CNF formula and a DNF formula implies their ti-equivalence but not vice versa.

We call a switch which maintains the size bounds from a DNF formula to a CNF formula a *DNF/CNF-approximator switch* and from a CNF formula to a DNF formula a *CNF/DNF-approximator switch*.

The proofs of the upper and lower bounds depend only on the size bounds k and r. The size bounds k and r are used to get an upper bound for the number of monomials of size larger than $r-1$ which are replaced by zero during a CNF/DNF-approximator switch and also to get an upper bound for the number of clauses of size larger than $k-1$ which are replaced by one during a DNF/CNF-approximator switch. The size bound r of the monomials is used to get an upper bound for the number of positive test inputs for which an error could be introduced by the replacement of one monomial by zero. The size bound k of the clauses is used to get an upper bound for the number of negative test inputs for which an error could be introduced by the replacement of one clause by one. The product of both upper bounds obtained with respect to a CNF/DNF-approximator switch gives us an upper bound for the number of positive test inputs for which an error could be introduced by a CNF/DNF-approximator switch. The product of both upper bounds obtained with respect to a DNF/CNF-approximator switch gives us an upper bound for the number of negative test inputs for which an error could be introduced by a DNF/CNF-approximator switch. No error with respect to a negative test input is introduced by a CNF/DNF-approximator switch and no error with respect to a positive test input is introduced by a DNF/CNF-approximator switch.

If the CNF representations are approximated then k is used to prove a lower bound for the number of test inputs for which $\mathcal{C}^k_{g_0}$ does not compute their values correctly. In the case that the DNF representations are approximated, r is used to prove a lower bound for the number of test inputs for which $\mathcal{D}^r_{g_0}$ does not compute their values correctly.

Now we are prepared to give a formal definition of a CNF-DNF-approximator for a monotone Boolean function.

A *CNF-DNF-approximator* $\mathcal{A}$ is a seven-tuple $(f, \Phi, (T_1, T_0), (\mathcal{S}, k, r), \mathcal{R}, (e_1, e_0), (d_1, d_0))$ such that:

a) f is the considered monotone Boolean function.

b) $\Phi \in \{\text{CNF, DNF}\}$ is the spezification which representation of the functions computed at the nodes of the network is approximated.

c) $T_1 \subseteq f^{-1}(1)$ and $T_0 \subseteq f^{-1}(0)$ are the sets of positive and negative test inputs.

d) $\mathcal{S}$ defines the sizes of the clauses and the monomials. k and r are the bounds for the sizes of the clauses and the monomials.

e) $\mathcal{R}$ defines the CNF/DNF-approximator switch and the DNF/CNF-approximator switch used by $\mathcal{A}$.

f) e_1 is an upper bound for the number of positive test inputs for which an error could be introduced by a CNF/DNF-approximator switch. e_0 is an upper bound for the number of negative test inputs for which an error could be introduced by a DNF/CNF-approximator switch.

g) $\mathcal{C}^k_{g_0}$ if $\Phi = \text{CNF}$ and $\mathcal{D}^r_{g_0}$ if $\Phi = \text{DNF}$, respectively contains for at least $d_1|T_1|$ positive test inputs a no monomial m_a with $m_a(a) = 1$ or contains for at least $d_0|T_0|$ negative test inputs b no clause d_b with $d_b(b) = 0$ where $0 < d_1, d_0 \leq 1$ are constants.

Given a CNF-DNF-approximator $\mathcal{A} = (f, \Phi, (T_1, T_0), (\mathcal{S}, k, r), \mathcal{R}, (e_1, e_0), (d_1, d_0))$ for a non-constant monotone Boolean function f and a monotone network β which computes at its output node g_0 the values $f(c)$ correctly for all test inputs $c \in T_1 \cup T_0$, we need a scheme for the use of $\mathcal{A}$ to construct the approximators corresponding to the nodes of β. Now we shall design such a scheme.

For an input node x_i, we define $\mathcal{D}^r_{x_i} := x_i$ and $\mathcal{C}^k_{x_i} := x_i$. For the definition of the approximators with respect to the gates, we consider the nodes in β in any topological order; i.e., when a gate g is considered then the approximators with respect to both direct predecessors h_1 and h_2 are defined. First, we shall consider the case that $\Phi = \text{CNF}$. According to the type of the gate g, we distinguish two cases:

Case 1: g is an $\vee$-gate.

Then we define
$$\mathcal{D}^r_g := \mathcal{D}^r_{h_1} \vee \mathcal{D}^r_{h_2}.$$
Since each monomial in $\mathcal{D}^r_{h_1}$ and in $\mathcal{D}^r_{h_2}$ has size less than r, all monomials in $\mathcal{D}^r_g$ have still size less than r. Moreover, since $\mathcal{D}^r_g$ is the same DNF formula as constructed by the network for the gate g before the approximation of the gate g, no error is introduced by the approximator $\mathcal{D}^r_g$.

We perform a DNF/CNF-approximator switch of $\mathcal{D}_g^r$ to obtain the approximator $\mathcal{C}_g^k$. If for an input $b \in T_0$, $d_b(g)$ is replaced by one then, instead of the clause d_b, the constant one is constructed at the output node g_0 by the subnetwork β_b such that an error with respect to b could be introduced by the approximator $\mathcal{C}_g^k$.

By construction, each clause in $\mathcal{C}_g^k$ contains a literal of every monomial in $\mathcal{D}_g^r$. Hence, each input which falsifies $\mathcal{C}_g^k$ falsifies $\mathcal{D}_g^r$ as well. Hence, no error with respect to a positive test input is introduced by $\mathcal{C}_g^k$.

Case 2: g is an $\wedge$-gate.

Then we define

$$\mathcal{C}_g^k := \mathcal{C}_{h_1}^k \wedge \mathcal{C}_{h_2}^k.$$

Since each clause in $\mathcal{C}_{h_1}^k$ and in $\mathcal{C}_{h_2}^k$ has size less than k, all clauses in $\mathcal{C}_g^k$ have still size less than k. Moreover, since $\mathcal{C}_g^k$ is the same CNF formula as constructed by the network for the gate g before the approximation of the gate g, no error would be introduced by the approximator $\mathcal{C}_g^k$.

For the eventual performance of a DNF/CNF-approximator switch with respect to a direct successor of the gate g, the approximator $\mathcal{D}_g^r$ is needed as well. We perform a CNF/DNF-approximator switch of $\mathcal{C}_g^k$ to obtain the approximator $\mathcal{D}_g^r$. If for an input $a \in T_1$, $m_a(g)$ is replaced by zero then, instead of the monomial m_a, the constant zero is constructed at the output node g_0 by the subnetwork β_a such that an error with respect to a could be introduced by the approximator $\mathcal{D}_g^r$.

By construction, each monomial in $\mathcal{D}_g^r$ contains a literal of every clause in $\mathcal{C}_g^k$. Hence, each input which satisfies $\mathcal{D}_g^r$ satisfies $\mathcal{C}_g^k$ as well. Hence, no error with respect to a negative test input is introduced by $\mathcal{D}_g^r$.

The scheme for the use of $\mathcal{A}$ in the case that $\Phi = \text{DNF}$ is the same as in the case that $\Phi = \text{CNF}$.

Before the definition of any approximator, β computes the value for each test input correctly. Hence, by Theorem 1, at the beginning, $\Phi(\beta)$ contains for each $a \in T_1$ a monomial m_a with $m_a(a) = 1$ and for each $b \in T_0$ a clause d_b with $d_b(b) = 0$. It is shown that after the definition of the approximators for the output node g_0, $\Phi(\beta)$ fails to contain a monomial m_a with $m_a(a) = 1$ for at least $d_1|T_1|$ positive test inputs a or fails to contain a clause d_b with $d_b(b) = 0$ for at least $d_0|T_0|$ negative test inputs b. At an $\wedge$-gate, exactly one CNF/DNF and no DNF/CNF-approximator switches and at an $\vee$-gate, exactly one DNF/CNF and no CNF/DNF-approximator switches are performed. For each gate in the monotone network β, the approximation introduces an error for at most e_0 test inputs in T_0 and for at most e_1 test inputs in T_1. A CNF-DNF-approximator $\mathcal{A} = (f, \Phi, (T_1, T_0), (\mathcal{S}, k, r), \mathcal{R}, (e_1, e_0), (d_1, d_0))$ proves a $\min\left\{\frac{d_1|T_1|}{e_1}, \frac{d_0|T_0|}{e_0}\right\}$ lower bound for $C_m(f)$.

4 Berg and Ulfberg's CNF-DNF-approximator for the clique function

For the understanding of the difficulties which occur if we intend the use of CNF-DNF-approximators for monotone Boolean functions on standard networks, the knowledge of a CNF-DNF-approximator for the clique function would be useful. Hence, we shall repeat the CNF-DNF-approximator for the clique function developed by Berg and Ulfberg [4]. Let CLIQUE(m, s) be the Boolean function of $n := \binom{m}{2}$ variables representing the edges of an undirected graph $G = (V, E)$ on m nodes. CLIQUE$(m, s)(x) = 1$ iff the corresponding graph G contains a clique of size s. In this section, f denotes the Boolean function CLIQUE(m, s).

The sets T_1 and T_0 of positive and negative test inputs should be defined in a way such that they are easily to analyze. Let $V' \subset V$ be a node set of size s. The graph consisting of the s-clique on the nodes in V' and $m-s$ isolated nodes in $V \setminus V'$ corresponds exactly to the prime implicant which contains the variables x_{uv} with $u, v \in V'$. A natural set T_1 of positive test inputs is the set of inputs corresponding exactly to the prime implicants of the function f; i.e., $T_1 := \text{PI}(f)$. Analogously, a natural set T_0 of negative test inputs could be the set of inputs corresponding exactly to the prime clauses of f; i.e., PC(f). The exact description of the set of graphs corresponding to the set PC(f) is more difficult than for PI(f). Hence, another subset of $f^{-1}(0)$ is more suitable for the definition of T_0. Let $h : V \to \{1, 2, \ldots, s-1\}$ be a colouring of the nodes in V by $s-1$ colours. The graph $G(h) = (V, E(h))$ corresponding to the colouring h contains all edges between two nodes in different colour classes and no edge between two nodes in the same colour class. The clause $d(h)$ corresponding to the colouring h contains exactly those variables x_{uv} with both nodes u and v are coloured with the same colour; i.e., $h(u) = h(v)$. Since each s-clique must contain at least two nodes which are in the same colour class, the clause $d(h)$ is an f-clause. For each clause $d(h)$ corresponding to a colouring h, the input $b \in \{0,1\}^n$ such that $b_{uv} = 0$ iff x_{uv} is a variable in $d(h)$ is called the $d(h)$*-input* of f. Note that exactly in the case that h uses all $s-1$ colours; i.e., $G(h)$ is a complete $(s-1)$-partite graph, the clause $d(h)$ is a prime clause of f. GC(f) denotes the set of all $d(h)$-inputs with respect to the colourings of V by $s-1$ colours. Different colourings can yield the same f-clause. Hence, $\mathrm{GC}(f)$ contains inputs corresponding to more than one colouring. To simplify the calculations, we shall consider such inputs as different. This means that the size of $\mathrm{GC}(f)$ is exactly the number $(s-1)^m$ of different colourings of m nodes by $s-1$ colours. We shall use the sets $T_1 := \mathrm{PI}(f)$ and $T_0 := \mathrm{GC}(f)$ of positive and negative test inputs.

The CNF-DNF-approximator of Berg and Ulfberg approximates the CNF formulas associated with the nodes in β. Now we shall spezify the size of a monomial and the size of a clause. We say that a monomial m or a clause d *touchs* a node $v \in V$ iff there is at least one variable in m or in d which corresponds to an edge in E with end node v. The *size of the monomial* m is the number of different nodes in V which are touched by m. For the definition of the size of a clause d, we consider the graph $G(d) = (V, E(d))$ where $E(d)$ contains exactly those edges which correspond to the variables in d. The *size of the clause* d is m minus the number of connected components in $G(d)$.

For the approximators $\mathcal{D}^r_g$ and $\mathcal{C}^k_g$, we use the values

$$r := \lfloor\sqrt{s}\rfloor \text{ and } k := \left\lfloor\frac{m}{8s}\right\rfloor.$$

Since $r = \lfloor\sqrt{s}\rfloor$, less than $\lfloor\sqrt{s}\rfloor$ different nodes in G can be touched by a monomial. Hence, the number of variables in such a monomial is bounded by $\frac{r^2}{2} \leq \frac{s}{2}$. Moreover, $k = \lfloor\frac{m}{8s}\rfloor$ implies that a graph corresponding to a clause has more than $m - \lfloor\frac{m}{8s}\rfloor$ connected components. If we mark in each connected component one node then less than $\frac{m}{8s}$ nodes remain unmarked. The number of different end nodes of the edges in such a graph is maximized if the number of connected components with exactly two nodes is maximized. Therefore, we obtain the maximum number of different end nodes if the unmarked nodes are distributed to pairwise different connected components. Hence, the number of different end nodes of the edges in such a graph is less than $2k \leq \frac{m}{4s}$. Therefore, a clause d touchs less than $\frac{m}{4s}$ nodes in V.

Next we shall describe the CNF/DNF-approximator switch developed by Berg and Ulfberg. Let $\mathcal{C}$ be a CNF formula which contains only clauses of size less than k. Our goal is to switch to a DNF formula $\mathcal{D}$ which contains only monomials of size less than r. First, a ti-equivalent DNF formula $\mathcal{D}'$ is constructed from $\mathcal{C}$. $\mathcal{D}$ is obtained from $\mathcal{D}'$ by the replacing of all monomials of size larger than $r-1$ by zero. $\mathcal{D}'$ will be constructed in a way such that at least one of the monomials in $\mathcal{D}'$ is satisfied by $a \in T_1$ iff all clauses in $\mathcal{C}$ are satisfied by a.

Let $d_1, d_2, \ldots, d_t$ be the clauses in $\mathcal{C}$ given in any fixed order and let d_0

be the empty clause. The construction of $\mathcal{D}'$ is organized by building a tree T as follows:

1. Each edge in T is labelled by a variable x_{uv} or has no label. With each node w in T we associate the monomial $m(w)$ which is obtained by the conjunction of the variables which are labels on the unique path from the root of T to w. T is constructed while *expanding* the clauses $d_0, d_1, d_2, \ldots, d_t$.

2. While expanding d_0, the root of T is created. The associated monomial is the empty monomial.

3. Suppose that w is a leaf which was created while expanding d_i.

Now we shall treat the clause d_{i+1} with respect to the leaf w. We call a variable x_{uv} *tight for a monomial* m iff both end nodes u and v are touched by m. We distinguish two cases:

Case 1: There is a variable x_{uv} in d_{i+1} which is tight for $m(w)$.

For each positive test input $a \in T_1$ which satisfies $m(w)$, both nodes u and v have to be contained in the clique which corresponds to the test input a. Hence, each such a test input satisfies the variable x_{uv} as well. We create only one son w' for w and label the edge (w, w') with x_{uv}.

Case 2: There is no variable in d_{i+1} which is tight for $m(w)$.

Then the variables in d_{i+1} separates into the following two sets:

$$\begin{aligned} Var_0 &:= \{x_{uv} \mid \text{both end nodes } u \text{ and } v \text{ are not touched by } m(w)\}, \\ Var_1 &:= \{x_{uv} \mid \text{exactly one of } u \text{ and } v \text{ is touched by } m(w)\}. \end{aligned}$$

First we shall consider the variables in Var_1. Let

$$V' := \{u \in V \mid u \text{ is not touched by } m(w) \text{ but there is } x_{uv} \in Var_1\}$$

and for each $u \in V'$ let

$$N(u) := \{v \in V \mid x_{uv} \in Var_1\}.$$

For each $u \in V'$ we choose any $v \in N(u)$ and create a son w_u of the node w. The edge (w, w_u) is labelled with the variable x_{uv} and we define that the edge (w, w_u) is touched by the node u. This suffices since two monomials touching the same nodes are submonomials of the same prime implicants of the function. Since the clause d_{i+1} touchs less than $2k$ nodes, less than

$$2k \leq \frac{m}{4s}$$

sons are created.

Now we shall consider the variables in Var_0. As long as there is an edge corresponding to a variable in Var_0 such that none of its two end nodes is chosen, we choose an end node u of such an edge and create a son w'_u for w. The corresponding edge (w, w'_u) obtains no label. We define that the edge (w, w'_u) is touched by the node u. Since d_{i+1} touchs less than $2k$ nodes, less than

$$2k \leq \frac{m}{4s}$$

sons are created. Let

$$V'' := \{u \in V \mid w'_u \text{ is created}\}$$

and for each $u \in V''$ let

$$N'(u) := \{v \in V \mid x_{uv} \in Var_0\}.$$

For each $u \in V''$ for each $v \in N'(u)$, we create a son w''_v of the node w'_u and label the edge (w'_u, w''_v) with the variable x_{uv}. We define that the node v touchs the edge (w'_u, w''_v). Again, since d_{i+1} touchs less than $2k$ nodes, less than

$$2k \leq \frac{m}{4s}$$

sons for w'_u are created.

After the construction of T, the monomials corresponding to the paths from the root of T to the leaves are the monomials in $\mathcal{D}'$. We obtain $\mathcal{D}$ from $\mathcal{D}'$ by the replacing of all monomials of size larger than $r - 1$ by zero. The following lemma bounds the number of positive test inputs for which an error could be introduced by a CNF/DNF-approximator switch.

Lemma 3 *Let $\mathcal{C}$ be a CNF formula which contains only clauses of size less than k. Let $\mathcal{D}$ be the DNF formula obtained by a CNF/DNF-approximator switch from $\mathcal{C}$. Then the number of test inputs in T_1 for which the approximator $\mathcal{D}$ could introduce an error is bounded by $\binom{m-r}{s-r}(\frac{m}{4s})^r$.*

Proof: By the construction of T there hold:

1. No node in T has more than $2k \leq \frac{m}{4s}$ sons.
2. When descending on an edge to a son from a node with more than one son, the number of nodes which are touched by the associated path increases by one. Hence, there are at most

$$\left(\frac{m}{4s}\right)^r$$

nodes in T such that the corresponding path from the root to the node touchs exactly r nodes in V.

Each path in T from the root to a leaf corresponding to a monomial in $\mathcal{D}'$ of size larger than $r-1$ contains a node w such that the path from the root to w touchs exactly r nodes in V. Note that r nodes are contained in $\binom{m-r}{s-r}$ s-cliques. Hence, the deletion of all monomials corresponding to a path from the root to a leaf which contains the node w could introduce an error for at most

$$\binom{m-r}{s-r}$$

positive test inputs $a \in T_1$.

Altogether, after the deletion of all monomials touching more than $r-1$ nodes, an error for at most

$$\binom{m-r}{s-r}\left(\frac{m}{4s}\right)^r$$

positive test inputs is introduced. □

Next we shall describe the DNF/CNF-approximator switch developed by Berg and Ulfberg. Let $\mathcal{D}$ be a DNF formula which contains only monomials of size less than r. Our goal is to switch to a CNF formula $\mathcal{C}$ which contains only clauses of size less than k. First, a ti-equivalent CNF formula $\mathcal{C}'$ is constructed from $\mathcal{D}$. $\mathcal{C}$ is obtained from $\mathcal{C}'$ by the replacing of all clauses of size larger than $k-1$ by one. $\mathcal{C}'$ will be constructed in a way such that at least one of the clauses in $\mathcal{C}'$ is falsified by $b \in T_0$ iff all monomials in $\mathcal{D}$ are falsified by b.

Let $m_1, m_2, \ldots, m_t$ be the monomials in $\mathcal{D}$ given in any fixed order and let m_0 be the empty monomial. The construction of $\mathcal{C}'$ is organized by building a tree T as follows:

1. Each edge in T is labelled by a variable x_{uv} . With each node w in T we associate the clause $d(w)$ which is obtained by the disjunction of the variables which are labels on the unique path from the root of T to w. T is constructed while *expanding* the monomials $m_0, m_1, m_2, \ldots, m_t$.

2. While expanding m_0, the root of T is created. The associated clause is the empty clause.

3. Suppose that w is a leaf which was created while expanding m_i.

Now we shall treat the monomial m_{i+1} with respect to the leaf w. A variable x_{uv} in m_{i+1} is called *good* iff both end nodes of the edge (u,v) are contained in the same connected component of the graph $G(d(w)) = (V, E(d(w))$. By construction, each $d(h)$-input b which falsifies $d(w)$ has the property that each connected component of $G(d(w))$ is contained in one colour class with respect to the colouring h. Hence, the $d(h)$-input b must falsify each good variable as well. We distinguish two cases:

Case 1: m_{i+1} contains a good variable x_{uv}.

Then a $G(h)$-input b which falsifies $d(w)$ also falsifies the variable x_{uv}. This implies $b_{uv} = 0$. Hence, it suffices to create one son w' of w and to label the edge (w,w') with the variable x_{uv}.

Case 2: m_{i+1} contains no good variable.

Then each variable x_{uv} in m_{i+1} connects two connected components of the graph $G(d(w))$. The leaf w obtains for each variable x_{uv} in m_{i+1} a new son w'. The edge (w,w') is labelled with the variable x_{uv}.

By construction, when decending on an edge to a son from a node with more than one son, the number of connected components of the associated graph decreases by one. Therefore, the size of the corresponding clause increases by one. Hence, there are at most k such nodes on a path from the root to a node with the property that the corresponding clause has exactly size k. Since each monomial in $\mathcal{D}$ contains at most $\frac{s}{2}$ variables, each node in T has at most degree $\frac{s}{2}$. Hence, there are at most

$$\left(\frac{s}{2}\right)^k$$

nodes in T such that the corresponding clause has exactly size k.

After the construction of T, the clauses corresponding to the paths from the root of T to the leaves are the clauses in $\mathcal{C}'$. We obtain $\mathcal{C}$ from $\mathcal{C}'$ by the replacing of all clauses of size larger than $k-1$ by one. The following lemma bounds the number of negative test inputs for which an error could be introduced by a DNF/CNF-approximator switch.

Lemma 4 *Let $\mathcal{D}$ be a DNF formula which contains only monomials of size less than r. Let $\mathcal{C}$ be the CNF formula obtained by a DNF/CNF-approximator switch from $\mathcal{D}$. Then the number of test inputs in T_0 for which the approximator $\mathcal{C}$ could introduce an error is bounded by $\left(\frac{s}{2}\right)^k (s-1)^{m-k}$.*

Proof: Each clause in $\mathcal{C}'$ of size larger than $k-1$ contains a subclause $d(w)$ which has exactly the size k where w is a node in T. This means that the corresponding graph $G(d(w))$ has exactly $m-k$ connected components.

Since each test input in T_0 is a $d(h)$-input for an appropriate colouring h of the node set V by $s-1$ colours, we need an upper bound for the number $d(h)$-inputs for which an error could be introduced by the approximator $\mathcal{C}$. Each $d(h)$-input which falsifies a clause in $\mathcal{C}'$ of size larger than $k-1$ must also falsify a subclause $d(w)$ of size k where w is a node in T. A $d(h)$-input b falsifies such a subclause $d(w)$ iff all nodes within the same connected component of $G(d(w))$ are coloured with the same colour. The number of different colourings of $m-k$ connected components by $s-1$ colours is $(s-1)^{m-k}$. Hence, there are at most

$$(s-1)^{m-k}$$

such $d(h)$-inputs. Since T contains at most $\left(\frac{s}{2}\right)^k$ nodes such that the corresponding clause has exactly the size k, an error for at most

$$\left(\frac{s}{2}\right)^k (s-1)^{m-k}$$

negative test inputs could be introduced by the approximator $\mathcal{C}$. □

Berg and Ulfberg [4] consider a monotone Boolean network β which computes the values for all test inputs in $T_1 \cup T_0$ correctly. They consider the approximator $\mathcal{C}^k_{g_0}$ of the output node g_0 of β and show that either $\mathcal{C}^k_{g_0}$ computes the value of all negative test inputs incorrectly or $\mathcal{C}^k_{g_0}$ computes the value of at least half of the positive test inputs incorrectly. For doing this, assume that there is $b \in T_0$ such that $\mathcal{C}^k_{g_o}(b) = 0$. Then $\mathcal{C}^k_{g_0}$ contains a clause d which is falsified by b. By construction, this clause d touchs less than $\frac{m}{4s}$ nodes of the graph. For each positive test input which satisfies $\mathcal{C}^k_{g_0}$, the corresponding prime implicant must touch one of these nodes. Every given node is part of the fraction $\frac{s}{m}$ of the possible s-cliques of a graph with m nodes. Hence, less than $\frac{m}{4s}$ nodes have a nonempty intersection with at most a fourth of the possible s-cliques. Therefore, the fraction of positive test inputs for which $\mathcal{C}^k_{g_0}$ computes the correct value is less than $\frac{1}{4}$. Since $\frac{3}{4} > \frac{1}{2}$, we have proved the following lemma.

Lemma 5 *Let β be a monotone Boolean network which computes the values for all test inputs in $T_1 \cup T_0$ correctly at its output node g_0. Then either $\mathcal{C}^k_{g_o}$ computes the value of all test inputs in T_0 incorrectly or $\mathcal{C}^k_{g_o}$ computes the value of at least half of the test inputs in T_1 incorrectly.*

Altogether, Berg and Ulfberg have constructed an approximator $\mathcal{A} = (f, \Phi, (T_1, T_0), (\mathcal{S}, k, r), \mathcal{R}, (e_1, e_0), (d_1, d_0))$ where

a) $f = \text{CLIQUE}(m, s)$,

b) $\Phi = \text{CNF}$,

c) $T_1 = \text{PI}(f)$ and $T_0 = \text{GC}(f)$,

d) $\mathcal{S}$ is the definition of the sizes as described above, $r := \lfloor\sqrt{s}\rfloor$ and $k := \lfloor\frac{m}{8s}\rfloor$,

e) $\mathcal{R}$ are the rules for the CNF/DNF- and DNF/CNF-approximator switches as described above,

f) $e_1 = \binom{m-r}{s-r}(\frac{m}{4s})^r$ and $e_0 = \left(\frac{s}{2}\right)^k (s-1)^{m-k}$, and

g) $d_1 = \frac{1}{2}$ and $d_0 = 1$.

Using Lemmas 3, 4 and 5, Berg and Ulfberg [4] have proved the following theorem.

Theorem 4 *Let $s \leq m^{\frac{2}{3}}$. Then $C_m(CLIQUE(m, s)) \geq 2^{\Omega(\sqrt{s})}$.*

Proof: We distinguish two cases:
Case 1: $\mathcal{C}_{g_0}^k$ computes the value of half of the positive test inputs incorrectly.

Because of Lemma 3, we obtain

$$\begin{aligned}
C_m(\text{CLIQUE}(m,s)) &\geq \frac{\binom{m}{s}}{2\binom{m-r}{s-r}(\frac{m}{4s})^r} \\
&= \frac{1}{2}\frac{m!(s-r)!(m-r-(s-r))!(4s)^r}{s!(m-s)!(m-r)!m^r} \\
&= \frac{1}{2}\frac{m!(s-r)!(4s)^r}{s!(m-r)!m^r} \\
&\geq \frac{1}{2}2^{(r+1)}\frac{(m-r)^r s^r}{s^r m^r} \\
&= (2 - \frac{2r}{m})^r \\
&> 2^{\Omega(r)} \\
&= 2^{\Omega(\sqrt{s})}.
\end{aligned}$$

Case 2: $\mathcal{C}_{g_0}^k$ computes the value of all negative test inputs incorrectly.

Because of Lemma 4, we obtain

$$\begin{aligned}
C_m(\text{CLIQUE}(m,s)) &\geq \frac{(s-1)^m}{(s-1)^{m-k}(\frac{s}{2})^k} \\
&= 2^k(\frac{s-1}{s})^k \\
&= 2^k(1-\frac{1}{s})^k \\
&= 2^{\Omega(\frac{m}{s})}.
\end{aligned}$$

Since $\frac{m}{s} \geq m^{\frac{1}{3}}$ for $s \leq m^{\frac{2}{3}}$, the assertion follows. $\Box$

5 Reduced CNF and DNF formulas

First we shall investigate the difficulties which occur if we intend the use of CNF-DNF-approximators defined for monotone Boolean functions on standard networks. For doing this, we shall use the function $f := \text{CLIQUE}(m, s)$. The characteristic function of the set $T_1 := \text{PI}(f)$ of positive test inputs can be computed in polynomial time. It suffices to check if $m - s$ nodes have degree zero and the other s nodes have degree $s - 1$. Hence, there exits a standard network of polynomial size for the characteristic function of T_1. Although this standard network does not compute the clique function, it computes the value of all test inputs in $T_1 \cup T_0$ correctly. Therefore, a CNF-DNF-approximator must use the fact that the approximator is used on a standard network which computes the clique function f at its output node.

In contrast to monotone Boolean networks, the monomials in the DNF representation and the clauses in the CNF representation of a function computed by a standard network contain negated variables. To elaborate the problems which arise because of the negated variables, we try to use Berg and Ulfberg's CNF-DNF-approximator for f on a standard network β which computes f at its output node g_0. This means that the sizes of the monomials and of the clauses only depend on their non-negated variables. What is the effect of the presence of negated variables on Berg and Ulfberg's analysis? For the development of the CNF/DNF-approximator switch and also of the DNF/CNF-approximator switch, a tree is constructed. If this would be done for the approximation of a standard network in an analogous manner, some edges in the tree must be labelled with a negated variable. A main argument in the proofs of Lemma 3 and Lemma 4 is that always when descending on an edge to a son from a node with more than one son, the size of the corresponding monomial or clause increases by one. But now, if the edge is labelled by a negated variable, this would not be the case. Hence, the proofs of Lemma 3 and Lemma 4 collapse. Moreover, the proof of Lemma 5 collapses as well. The clause d in $\mathcal{C}^k_{g_0}$ which falsifies the test input $b \in T_0$ can contain any number of negated variables. Each positive test input which satisfies $\mathcal{C}^k_{g_0}$ must satisfy one of the literals in d. But this can also be one of the negated variables in d. Hence, no positive test input can be excluded to be computed correctly by the approximator $\mathcal{C}^k_{g_0}$.

It seems that a CNF-DNF-approximator has also to approximate the negated variables. This would be a very difficult task. But Theorem 1 opens another way. For each positive test input a, $\text{DNF}_\beta(g_0)$ and also $\text{CNF}_\beta(g_0)$ contain an implicant m_a such that $m_a(a) = 1$. Moreover, for each negative test input b, they contain an f-clause d_b such that $d_b(b) = 0$. Since the clique function is monotone, after the removal of the negated variables in m_a and in d_b, we obtain a monomial m'_a which is still an implicant of f and a clause d'_b which is still an f-clause. Obviously, $m'_a(a) = 1$ and $d'_b(b) = 0$. This suggests the following approach:

Apply a transformation which eliminates the negated variables in the monomials of the DNF representation and in the clauses of the CNF representation of the functions computed at the nodes of the given standard network first and then approximate the resulting *reduced* CNF representations or the resulting *reduced* DNF representations.

To get such a transformation, we also introduce some additional rules for the construction of reduced DNF and reduced CNF formulas. To eliminate the negated variables from the monomials, during the construction of a reduced DNF formula, a non-empty monomial m which contains only non-negated variables always absorbs the negated variables. To eliminate the negated variables from the clauses, during the construction of a reduced CNF formula, a non-empty clause d which contains only non-negated variables always absorbs the negated variables. This means that during the construction of a reduced DNF formula, the rule

$$m \wedge m' = m,$$

where m is non-empty and contains only non-negated variables and m' contains only negated variables, is applied. During the construction of a reduced CNF formula, the rule

$$d \vee d' = d$$

where d is non-empty and contains only non-negated variables and d' contains only negated variables is applied.

By construction, the DNF and CNF representations of the functions computed at the nodes of a standard network can also contain trivial monomials or trivial clauses. Because of the absorbtion of the negated variables, a trivial monomial or a trivial clause would be transformed into a non-trivial monomial or a non-trivial clause. Since a trivial monomial or a trivial clause contains for at least one variable both literals, a trivial monomial or a trivial clause contains for each positive test input a a literal which is fulfilled by a and for each negative test input b a literal which is falsified by b. After the tranformation, the resulting non-trivial clause can lose this property such that the transformation would have a disturbing side effect. To remove this side effect, we define a further operator R, which, applied to a DNF formula, replaces all monomials which originate from a trivial monomial by zero. Applied to a CNF formula, R replaces all clauses which originate from a trivial clause by one.

By construction, each non-empty monomial in a reduced DNF formula and each non-empty clause in a reduced CNF formula contains only non-negated variables or only negated variables. The term for a monomial or a clause is overlined iff it contains only negated variables. Hence, each reduced DNF formula α and each reduced CNF formula γ can be represented in the following way:

$$\alpha = \bigvee_{i=1}^{t} m_i \;\vee\; \bigvee_{j=1}^{t'} \overline{m}_j \quad \text{and} \quad \gamma = \bigwedge_{i=1}^{t} d_i \;\wedge\; \bigwedge_{j=1}^{t'} \overline{d}_j,$$

where $t = 0$ or $t' = 0$ if the corresponding subformula is empty.

Now we are prepared to describe the *reduced DNF and CNF formulas* constructed by a standard network. Starting at the input nodes, the standard network β constructs the reduced DNF representations DNF'$_\beta(g)$ of the functions $\mathrm{res}_\beta(g)$ in the following way:

1. If g is an input node with $\mathrm{op}(g) = x_i$ or $\mathrm{op}(g) = \neg x_i$ then

$$\text{DNF'}_\beta(g) := \mathrm{op}(g).$$

2. If g is an $\vee$-gate with $\mathrm{pred}(g) = \{h_1, h_2\}$ then

$$\text{DNF'}_\beta(g) := \text{DNF'}_\beta(h_1) \vee \text{DNF'}_\beta(h_2).$$

3. If g is an $\wedge$-gate with $\mathrm{pred}(g) = \{h_1, h_2\}$, DNF'$_\beta(h_1) = \bigvee_{i=1}^{t_1} m_i \;\vee\; \bigvee_{k=1}^{t_1'} \overline{m}_k$ and DNF'$_\beta(h_2) = \bigvee_{j=1}^{t_2} m_j' \;\vee\; \bigvee_{l=1}^{t_2'} \overline{m'}_l$ then we distinguish four subcases:

   a) $t_1' = 0$ and $t_2' = 0$. Then

$$\alpha := \bigvee_{i=1}^{t_1} \bigvee_{j=1}^{t_2} (m_i \wedge m_j').$$

   b) $t_1' > 0$ and $t_2' = 0$. Then

$$\alpha := \bigvee_{j=1}^{t_2} m_j' \;\vee\; \bigvee_{i=1}^{t_1} \bigvee_{j=1}^{t_2} (m_i \wedge m_j').$$

   c) $t_1' = 0$ and $t_2' > 0$. Then

$$\alpha := \bigvee_{i=1}^{t_1} m_i \;\vee\; \bigvee_{i=1}^{t_1} \bigvee_{j=1}^{t_2} (m_i \wedge m_j').$$

d) $t'_1 > 0$ and $t'_2 > 0$. Then

$$\alpha := \bigvee_{i=1}^{t_1} m_i \;\vee\; \bigvee_{j=1}^{t_2} m'_j \;\vee\; \bigvee_{i=1}^{t_1}\bigvee_{j=1}^{t_2}(m_i \wedge m'_j) \;\vee\; \bigvee_{k=1}^{t'_1}\bigvee_{l=1}^{t'_2}(\overline{m}_k \wedge \overline{m'}_l).$$

In all subcases, we apply the operator R to α to obtain DNF'$_\beta(g)$; i.e.,

$$\text{DNF'}_\beta(g) := R(\alpha).$$

Starting at the input nodes, the standard network β constructs the reduced CNF representations CNF'$_\beta(g)$ of the functions $\text{res}_\beta(g)$ in the following way:

1. If g is an input node with $\text{op}(g) = x_i$ or $\text{op}(g) = \neg x_i$ then

$$\text{CNF'}_\beta(g) := \text{op}(g).$$

2. If g is an $\wedge$-gate with $\text{pred}(g) = \{h_1, h_2\}$ then

$$\text{CNF'}_\beta(g) := \text{CNF'}_\beta(h_1) \wedge \text{CNF'}_\beta(h_2).$$

3. If g is an $\vee$-gate with $\text{pred}(g) = \{h_1, h_2\}$, $\text{CNF'}_\beta(h_1) = \bigwedge_{i=1}^{t_1} d_i \wedge \bigwedge_{k=1}^{t'_1} \overline{d}_k$ and $\text{CNF'}_\beta(h_2) = \bigwedge_{j=1}^{t_2} d'_j \wedge \bigwedge_{l=1}^{t'_2} \overline{d'}_l$ then we distinguish four subcases:

a) $t'_1 = 0$ and $t'_2 = 0$. Then

$$\gamma := \bigwedge_{i=1}^{t_1}\bigwedge_{j=1}^{t_2}(d_i \vee d'_j).$$

b) $t'_1 > 0$ and $t'_2 = 0$. Then

$$\gamma := \bigwedge_{j=1}^{t_2} d'_j \;\wedge\; \bigwedge_{i=1}^{t_1}\bigwedge_{j=1}^{t_2}(d_i \vee d'_j).$$

c) $t'_1 = 0$ and $t'_2 > 0$. Then

$$\gamma := \bigwedge_{i=1}^{t_1} d_i \;\wedge\; \bigwedge_{i=1}^{t_1}\bigwedge_{j=1}^{t_2}(d_i \vee d'_j).$$

d) $t'_1 > 0$ and $t'_2 > 0$. Then

$$\gamma := \bigwedge_{i=1}^{t_1} d_i \;\wedge\; \bigwedge_{j=1}^{t_2} d'_j \;\wedge\; \bigwedge_{i=1}^{t_1}\bigwedge_{j=1}^{t_2}(d_i \vee d'_j) \;\wedge\; \bigwedge_{k=1}^{t'_1}\bigwedge_{l=1}^{t'_2}(\overline{d}_k \vee \overline{d'}_l).$$

In all subcases, we apply the operator R to γ to obtain CNF'$_\beta(g)$; i.e.,

$$\text{CNF'}_\beta(g) := R(\gamma).$$

Alternatively, we can obtain the reduced CNF and DNF formulas of the standard network β in the following way: The CNF and DNF representations of the functions computed at the nodes of β are constructed first. Then all trivial monomials are replaced by zero and all trivial clauses are replaced by one. Finally, the absorbtion rules are applied to these formulas obtaining for each node g of β the reduced formulas CNF'$_\beta(g)$ and DNF'$_\beta(g)$.

Obviously, we obtain the same reduced CNF and DNF representations of the functions computed at the nodes of the standard network β if we construct the CNF and DNF representations first and then applying the removing and absorbtion rules or if we construct the reduced CNF and DNF representations as described above directly. This can be proved by induction. The following theorem characterizes the reduced CNF and DNF formulas constructed at the output node of a standard network which computes a non-constant monotone Boolean function.

Theorem 5 *Let $f \in \mathcal{B}_n$ be a non-constant monotone Boolean function. Let β be a standard network which computes f at the output node g_0. Then for DNF'$_\beta(g_0)$ and for CNF'$_\beta(g_0)$, the following hold:*

a) All monomials contained in DNF'$_\beta(g_0)$ are implicants of f. For each $a \in f^{-1}(1)$, DNF'$_\beta(g_0)$ contains an implicant m'_a of f such that $m'_a(a) = 1$. For each $b \in f^{-1}(0)$, DNF'$_\beta(g_0)$ contains an f-clause d'_b such that $d'_b(g) = 0$.

b) All clauses contained in CNF'$_\beta(g_0)$ are f-clauses. For each $b \in f^{-1}(0)$, CNF'$_\beta(g_0)$ contains an f-clause d'_b such that $d'_b(b) = 0$. For each $a \in f^{-1}(1)$, CNF'$_\beta(g_0)$ contains an implicant m'_a of f such that $m'_a(a) = 1$.

Proof: By the removing rules, each trivial monomial in DNF$_\beta(g_0)$ is replaced by zero. Hence, each monomial m' in DNF'$_\beta(g_0)$ is obtained by an application of the absorbtion rule to an implicant $m = m'm''$ of f. Since m' is still an implicant of f, all monomials in DNF'$_\beta(g_0)$ are implicants of f.

By Theorem 1a, $\mathrm{DNF}_\beta(g_0)$ contains for each $a \in f^{-1}(1)$ an implicant m_a of f such that $m_a(a) = 1$. We can write $m_a = m'_a m''_a$ where m'_a contains only non-negated variables and m''_a is empty or contains only negated variables. Hence, by the absorbtion rule, DNF'$_\beta(g_0)$ contains the monomial m'_a. Since f is monotone, the monomial m'_a is still an implicant of f.

This shows also that DNF'$_\beta(g_0)$ computes f. Note that all monomials in DNF'$_\beta(g_0)$ contain only non-negated variables. By definition, DNF'$_\beta(g_0)$ contains exactly those clauses which are contained in that CNF formula γ which results by a DNF/CNF-switch of DNF'$_\beta(g_0)$. By Lemma 1, γ computes f as well. Hence, by Theorem 1, γ contains for each $b \in f^{-1}(0)$ an f-clause d'_b such that $d'_b(b) = 0$.

This proves part a) of the theorem. Analogously, we can prove part b) of the theorem. □

In contrast to monotone networks, we need that the standard network β computes the monotone function under consideration. Otherwise, we cannot ensure that the reduced CNF and DNF representations of $\mathrm{res}_\beta(g_0)$ computes the values for all test inputs correctly.

6 CNF-DNF-Approximators for Monotone Boolean Functions used on Standard Networks

Given a a standard network β which computes a non-constant monotone Boolean function f at its output node g_0 and a CNF-DNF-approximator $\mathcal{A} = (f, \Phi,\ (T_1, T_0), (\mathcal{S}, k, r), R, (e_1, e_0), (d_1, d_0))$, we wish to use $\mathcal{A}$ on β. More precisely, we wish to approximate the reduced Φ formulas of the functions computed at the nodes of β. For doing this, we define the *current front* in the same way as for monotone networks. Analogously to monotone networks, we specify when an error is introduced by an approximator. The size of a monomial or a clause consisting of only negated variables is defined to be zero. We need a scheme for the use of $\mathcal{A}$ to construct the approximators corresponding to the nodes of β. Our goal is to design such a scheme.

For an input node g with $\mathrm{op}(g) = x_i$ or $\mathrm{op}(g) = \neg x_i$, we define $\mathcal{D}^r_{x_i} := \mathrm{op}(g)$ and $\mathcal{C}^k_{x_i} := \mathrm{op}(g)$. For the definition of the approximators for the gates, we consider the nodes in β in a topological order; i.e., when a gate g is considered then the approximators of both direct predecessors h_1 and h_2 are defined. First we shall consider the case that $\Phi = \mathrm{CNF}$. According to the type of g, we distinguish two cases.

Case 1: g is an $\vee$-gate.

Let

$$\mathcal{C}^k_{h_1} = \gamma_1 \wedge \gamma'_1 \text{ where } \gamma_1 = \bigwedge_{i=1}^{t_1} d_i \text{ and } \gamma'_1 = \bigwedge_{k=1}^{t'_1} \overline{d}_k,$$

$$\mathcal{C}^k_{h_2} = \gamma_2 \wedge \gamma'_2 \text{ where } \gamma_2 = \bigwedge_{j=1}^{t_2} d'_j \text{ and } \gamma'_2 = \bigwedge_{l=1}^{t'_2} \overline{d'}_l,$$

$$\gamma = \bigwedge_{i=1}^{t_1} \bigwedge_{j=1}^{t_2} (d_i \vee d'_j) \text{ and } \gamma' = \bigwedge_{k=1}^{t'_1} \bigwedge_{l=1}^{t'_2} (\overline{d}_k \vee \overline{d'}_l).$$

Before the approximation of the gate g, a clause d in CNF'$_\beta(g)$ can use a clause in γ'_1 or in γ'_2 or no clause in γ'_1 or in γ'_2. Since these situations have to be treated differently, the construction of the approximator separates into two steps. During the first step, a CNF formula $\mathcal{C}'_g$ containing exactly those clauses in CNF'$_\beta(g)$ which use at least one clause in γ'_1 or in γ'_2 is constructed. In the second step, a CNF formula $\mathcal{C}''_g$ containing the clauses which use no clause in γ'_1 or in γ'_2 is constructed. During the construction of the approximators, we have to apply the operator R for the removal of clauses which stem from trivial clauses. Finally, the approximator $\mathcal{C}^k_g$ is obtained by the conjunction of both constructed CNF formulas $\mathcal{C}'_g$ and $\mathcal{C}''_g$.

Step 1:

We define

$$\mathcal{C}'_g := \begin{cases} R(\gamma_2) & t'_1 > 0 \text{ and } t'_2 = 0, \\ R(\gamma_1) & t'_1 = 0 \text{ and } t'_2 > 0, \\ R(\gamma_1 \wedge \gamma_2 \wedge \gamma') & t'_1 > 0 \text{ and } t'_2 > 0, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Obviously, all clauses in $\mathcal{C}'_g$ have size less than k. Furthermore, $\mathcal{C}'_g$ contains still all clauses contained in CNF'$_\beta(g)$ before the approximation of the gate g which use a clause in γ'_1 or in γ'_2.

Step 2:

We have to realize $R(\gamma)$. Instead of doing this directly, we shall perform a DNF/CNF-approximator switch using an appropriate DNF formula which contains only monomials of size less than r. Assume that we have an approximator $\mathcal{D}^r_{h_1}$ for γ_1 and an approximator $\mathcal{D}^r_{h_2}$ for γ_2. Then we define

$$\mathcal{D}'_g := \mathcal{D}^r_{h_1} \vee \mathcal{D}^r_{h_2}.$$

By construction, $\mathcal{D}'_g$ is a DNF formula where all monomials have size less than r. Then $\mathcal{C}''_g$ is obtained by performing a DNF/CNF-approximator switch using the DNF formula $\mathcal{D}'_g$ and applying the operator R to the resulting CNF formula.

Now we obtain the approximator $\mathcal{C}^k_g$ by

$$\mathcal{C}^k_g := \begin{cases} \mathcal{C}'_g \wedge \mathcal{C}''_g & \text{if } \mathcal{C}'_g \text{ and } \mathcal{C}''_g \text{ are defined,} \\ \mathcal{C}'_g & \text{if only } \mathcal{C}'_g \text{ is defined,} \\ \mathcal{C}''_g & \text{if only } \mathcal{C}''_g \text{ is defined.} \end{cases}$$

By construction, all clauses in $\mathcal{C}^k_g$ have size less than k. Let

$$\mathcal{C}^k_g = \gamma \wedge \gamma' \text{ where } \gamma = \bigwedge_{i=1}^{t} d_i \text{ and } \gamma' = \bigwedge_{k=1}^{t'} \overline{d}_k.$$

For defining the approximators of the direct successors of the gate g, we need an approximator $\mathcal{D}^r_g$ for γ as well. The approximator $\mathcal{D}^r_g$ is obtained by performing a CNF/DNF-approximator switch using the CNF formula γ.

During the construction of the approximators $\mathcal{C}^k_g$ and $\mathcal{D}^r_g$, one DNF/CNF-approximator switch and one CNF/DNF-approximator switch are performed. By the structure of the approximator $\mathcal{A}$, an error is introduced for at most e_0 negative and at most e_1 positive test inputs.

Case 2: g is an $\wedge$-gate.

Then we define

$$\mathcal{C}^k_g := \mathcal{C}^k_{h_1} \wedge \mathcal{C}^k_{h_2}.$$

By construction, all clauses in $\mathcal{C}^k_g$ have size less than k. Let

$$\mathcal{C}^k_g = \gamma \wedge \gamma' \text{ where } \gamma = \bigwedge_{i=1}^{t} d_i \text{ and } \gamma' = \bigwedge_{k=1}^{t'} \overline{d}_k.$$

For defining the approximators of the direct successors of the gate g, we need an approximator $\mathcal{D}^r_g$ for γ as well. The approximator $\mathcal{D}^r_g$ is obtained by performing a CNF/DNF-approximator switch using the CNF formula γ.

During the construction of the approximators $\mathcal{C}^k_g$ and $\mathcal{D}^r_g$, one CNF/DNF-approximator switch and no DNF/CNF-approximator switch is performed. By the structure of the approximator $\mathcal{A}$, an error is introduced for no negative and at most e_1 positive test inputs.

It remains to consider the case that Φ = DNF. Remember that for monotone Boolean networks, both schemes for Φ = CNF and for Φ = DNF formulas are identically. This is not the case for standard networks. But the constructions of both schemes are dual. Although the construction of the scheme for $\Phi = \text{DNF}$ is straightforward, we shall present the scheme now. According to the type of g, we distinguish two cases.

Case 1: g is an $\wedge$-gate.

Let

$$\mathcal{D}^r_{h_1} = \alpha_1 \vee \alpha'_1 \;\text{ where }\; \alpha_1 = \bigvee_{i=1}^{t_1} m_i \text{ and } \alpha'_1 = \bigvee_{k=1}^{t'_1} \overline{m}_k,$$

$$\mathcal{D}^r_{h_2} = \alpha_2 \vee \alpha'_2 \;\text{ where }\; \alpha_2 = \bigvee_{j=1}^{t_2} m'_j \text{ and } \alpha'_2 = \bigvee_{l=1}^{t'_2} \overline{m'}_l,$$

$$\alpha = \bigvee_{i=1}^{t_1} \bigvee_{j=1}^{t_2} (m_i \wedge m'_j) \;\text{ and }\; \alpha' = \bigvee_{k=1}^{t'_1} \bigvee_{l=1}^{t'_2} (\overline{m}_k \wedge \overline{m'}_l).$$

Again, the construction of the approximator separates into two steps. During the first step, a DNF formula $\mathcal{D}'_g$ containing exactly those monomials in DNF'$_\beta(g)$ which use at least one monomial in α'_1 or in α'_2 is constructed. In the second step, a DNF formula $\mathcal{D}''_g$ containing the monomials which use no monomial in α'_1 or in α'_2 is constructed. Again, we have to apply the operator R. Finally, the approximator $\mathcal{D}^r_g$ is obtained by the disjunction of both constructed DNF formulas $\mathcal{D}'_g$ and $\mathcal{D}''_g$.

Step 1:

We define

$$\mathcal{D}'_g := \begin{cases} R(\alpha_2) & t'_1 > 0 \text{ and } t'_2 = 0, \\ R(\alpha_1) & t'_1 = 0 \text{ and } t'_2 > 0, \\ R(\alpha_1 \vee \alpha_2 \vee \alpha') & t'_1 > 0 \text{ and } t'_2 > 0, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Obviously, all monomials in $\mathcal{D}'_g$ have size less than r. Furthermore, $\mathcal{D}'_g$ contains still all monomials contained in DNF'$_\beta(g)$ before the approximation of the gate g which use a monomial in α'_1 or in α'_2.

Step 2:

We have to realize $R(\alpha)$. Instead of doing this directly, we shall perform a CNF/DNF-approximator switch using an appropriate CNF formula which contains only clauses of size less than k. Assume that we have an approximator $\mathcal{C}^k_{h_1}$ for α_1 and an approximator $\mathcal{C}^k_{h_2}$ for α_2. Then we define

$$\mathcal{C}'_g := \mathcal{C}^k_{h_1} \wedge \mathcal{C}^k_{h_2}.$$

By construction, $\mathcal{C}'_g$ is a CNF formula where all clauses have size less than k. Then $\mathcal{D}''_g$ is obtained by performing a CNF/DNF-approximator switch using the CNF formula $\mathcal{C}'_g$ and applying the operator R to the resulting DNF formula.

Now we obtain the approximator $\mathcal{D}^r_g$ by

$$\mathcal{D}^r_g := \begin{cases} \mathcal{D}'_g \vee \mathcal{D}''_g & \text{if } \mathcal{D}'_g \text{ and } \mathcal{D}''_g \text{ are defined,} \\ \mathcal{D}'_g & \text{if only } \mathcal{D}'_g \text{ is defined,} \\ \mathcal{D}''_g & \text{if only } \mathcal{D}''_g \text{ is defined.} \end{cases}$$

By construction, all monomials in $\mathcal{D}^r_g$ have size less than r. Let

$$\mathcal{D}^r_g = \alpha \vee \alpha' \;\text{ where }\; \alpha = \bigvee_{i=1}^{t} m_i \text{ and } \alpha'_1 = \bigvee_{k=1}^{t'} \overline{m}_k.$$

For defining the approximators of the direct successors of the gate g, we need an approximator $\mathcal{C}^k_g$ for α as well. The approximator $\mathcal{C}^k_g$ is obtained by performing a DNF/CNF-approximator switch using the DNF formula α.

During the construction of the approximators, one CNF/DNF- and one DNF/CNF-approximator switch are performed. By the structure of the approximator $\mathcal{A}$, an error is introduced for at most e_1 positive and at most e_0 negative test inputs.

Case 2: g is an $\vee$-gate.

Then we define

$$\mathcal{D}^r_g := \mathcal{D}^r_{h_1} \vee \mathcal{D}^r_{h_2}.$$

By construction, all monomials in $\mathcal{D}^r_g$ have size less than r. Let

$$\mathcal{D}^r_g = \alpha \vee \alpha' \;\text{ where }\; \alpha = \bigvee_{i=1}^{t} m_i \text{ and } \alpha'_1 = \bigvee_{k=1}^{t'} \overline{m}_k.$$

For defining the approximators of the direct successors of the gate g, we need an approximator $\mathcal{C}^k_g$ for α as well. The approximator $\mathcal{C}^k_g$ is obtained by performing a DNF/CNF-approximator switch using the DNF formula α.

During the construction of the approximators $\mathcal{D}_g^r$ and $\mathcal{C}_g^k$, one DNF/CNF- and no CNF/DNF-approximator switch are performed. By the structure of the approximator $\mathcal{A}$, an error is introduced for no positive and at most e_0 negative test inputs.

By Theorem 5, before the definition of any approximator, $\Phi(\beta)$ contains for each $a \in T_1$ a monomial m_a with $m_a(a) = 1$ and for each input $b \in T_0$ a clause d_b with $d_b(b) = 0$. By construction and Theorem 5, the approximators $\mathcal{C}_{g_0}^k$ and $\mathcal{D}_{g_0}^r$ do not contain any negated variable. Furthermore, all monomials in $\mathcal{D}_{g_0}^r$ have size less than r and all clauses in $\mathcal{C}_{g_0}^k$ have size less than k. Hence, by the definition of the approximators and the structure of $\mathcal{A}$, the approximator $\Phi(\beta)$ contains for at least $d_1|T_1|$ positive test inputs a no monomial m_a with $m_a(a) = 1$ or contains for at least $d_0|T_0|$ negative test inputs b no clause d_b with $d_b(b) = 0$. Since for each gate in the standard network β, the approximation introduces an error for at most e_0 test inputs in T_0 and for at most e_1 test inputs in T_1, a $\min\left\{\frac{d_1|T_1|}{e_1}, \frac{d_0|T_0|}{e_0}\right\}$ lower bound for $C_{st}(f)$ is proved.

Altogether, we have proved the following theorem.

Theorem 6 *Let $f \in \mathcal{B}_n$ be any monotone Boolean function. Assume that there is a CNF-DNF-approximator $\mathcal{A}$ which can be used to prove a lower bound for $C_m(f)$. Then $\mathcal{A}$ can also be used to prove the same lower bound for $C_{st}(f)$.*

7 Applications

Improving the lower bound of Razborov [11], Alon and Boppana [1] have proved for $s \leq m^{2/3}$ a $2^{\Omega(\sqrt{s})}$ lower bound for the monotone complexity of CLIQUE(m, s). Using a CNF-DNF-approximator, Berg and Ulfberg [4] have proved the same lower bound. By an application of Theorem 6, we obtain the following theorem.

Theorem 7 *Let $s \leq m^{\frac{2}{3}}$. Then $C_{st}(CLIQUE(m, s)) \geq 2^{\Omega(\sqrt{s})}$.*

Andreev [3] was the first who could prove an exponential lower bound for the monotone complexity of a Boolean function in NP. Andreev's function is the characteristic function POLY(q, s) of the following problem:

For a prime power $q \geq 2$ let $GF(q)$ denote the finite field with q elements. Let $G = (A, B, E)$ be a bipartite graph where $A := GF(q)$ and $B := GF(q)$. For given q and s, the problem is to decide whether there exists a polynomial

p over $GF(q)$ of degree at most $s-1$ such that for all $i \in A$ there hold $(i, p(i)) \in E$.

POLY(q,s) is a monotone Boolean function of $n := q^2$ variables. For $s = \frac{1}{2}n^{1/8}/\sqrt{\ln n} - 1$, Andreev has obtained a $2^{\Omega(n^{1/8}/\sqrt{\ln n})}$ lower bound for the monotone complexity of POLY(q,s). Alon and Boppana [1] have improved this for $s \leq \frac{1}{2}\sqrt{q/\ln q}$ to $q^{\Omega(s)}$. Therefore, after setting $s := \frac{1}{2}\sqrt{q/\ln q}$, we obtain a $2^{\Omega(n^{1/4}/\sqrt{\ln n})}$ lower bound. Using a CNF-DNF-approximator, Berg and Ulfberg [4] have proved the same lower bound. By an application of Theorem 6, we obtain the following theorem.

Theorem 8 *Let $s \leq \frac{1}{2}\sqrt{q/\ln q}$. Then $C_{st}(POLY(q,s)) \geq q^{\Omega(s)}$. For $s := \frac{1}{2}\sqrt{q/\ln q}$ there holds $C_{st}(POLY(q,s)) \geq 2^{\Omega(n^{1/4}/\sqrt{\ln n})}$.*

Since the languages corresponding to both functions are contained in NP, we obtain the following corollary.

Corollary 1 *Let P be the set of languages accepted in polynomial time by a deterministic Turing machine and let NP be the set of languages accepted in polynomial time by a nondeterministic Turing machine. Then $P \neq NP$.*

8 Negations in Boolean Networks

We have shown that every CNF-DNF-approximator developed to prove a certain lower bound for the monotone complexity of a considered monotone function can be used to prove the same lower bound for its standard complexity. This explains why Berg and Ulfberg [4] could not find a CNF-DNF-approximator for proving Razborov's lower bound for the perfect matching function. This means that CNF-DNF-approximators filter out anything what can be done with help of negations. For the perfect matching function, it would be interesting to get CNF-DNF-approximators yielding a lower bound for its non-monotone complexity which would exclude the existence of for example an $O(m^2 \log^2 m)$ algorithm. But yet, Boolean networks are related to Turing machine computations and not to computations of a RAM. How large must be such a lower bound to exclude an algorithm of certain complexity on a RAM?

Further questions with respect to fundamental functions remain to be open. Can we multiply two integers in linear time or can we prove an $\Omega(n \log n)$ lower bound for the non-monotone complexity of the multiplication of two n-bit numbers? Is FFT the best what we can do with respect to polynomial multiplication? What is the complexity of the Boolean matrix multiplication? It is my opinion that the explore of the power of negations remains to be one of the greatest challenges in Theoretical Computer Science.

References

[1] Alon, N., Boppana, R. B.: The monotone circuit complexity of Boolean functions, *Combinatorica* **7** (1987), 1–22.

[2] Amano, K., Maruoka, A.: The potential of the approximation method, *SIAM J. Comput.* **33** (2004), 433–447.

[3] Andreev, A. E.: On a method for obtaining lower bounds for the complexity of individual monotone functions, *Soviet Math. Dokl.* **31** (1985), 530–534.

[4] Berg, C., Ulfberg, S.: Symmetric approximation arguments for monotone lower bounds without sunflowers, *Comput. Complex.* **8** (1999), 1–20.

[5] Blum, N.: On negations in Boolean networks, in Albers, S., Alt, H., Näher, S. (eds.): *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, LNCS **5760** (2009), 18–29.

[6] Haken, A.: Counting bottlenecks to show monotone $P \neq NP$, *Proc. 36th FOCS* (1995), 36–40.

[7] Harnik, D., Raz, R.: Higher lower bounds on monotone size, *Proc. 32nd STOC* (2000), 191–201.

[8] Jukna, S.: Combinatorics of monotone computations, *Combinatorica* **19** (1999), 65–85.

[9] Jukna, S.: *Boolean Function Complexity: Advances and Frontiers*, Springer 2012.

[10] Karchmer, M.: On proving lower bounds for circuit size, *Proc. 8th Structure in Complexity Theory* (1993), 112–118.

[11] Razborov, A. A.: Lower bounds on the monotone complexity of some Boolean functions, *Soviet Math. Dokl.* **31** (1985), 354–357.

[12] Razborov, A. A.: A lower bound on the monotone network complexity of the logical permanent, *Math. Notes Acad. Sci. USSR* **37** (1985), 485–493.

[13] Razborov, A. A.: On the method of approximation, *Proc. 21st STOC* (1989), 167–176.

[14] Razborov A.A., Rudich, S.: Natural proofs, *JCSS* **55** (1997), 24–35.

[15] Savage, J. E.: *Models of Computation: Exploring the Power of Computing*, Addison-Wesley 1998.

[16] Tardos, É., The gap between monotone and non-monotone circuit complexity is exponential, *Combinatorica* **8**, 141–142.