TOPIC 6
The Mystery of Negations

(Related sections: 9, 10.6)

Summary: Monotone boolean functions can be computed by circuits with no negation gates. At the cost of a small increase in size, *every* boolean function on n variables can be computed using only $\log n$ negation gates. There are explicit *monotone* functions that *can* be computed by polynomial-size circuit using $\log n$ negations, but every circuit with $(1-\varepsilon)\log n$ negations must have exponential size.

Let us consider boolean circuits with And, Or and Not gates. Such a circuit C is just a sequence $C = (g_1, \dots, g_t)$ of boolean functions $g_i : \{0,1\}^n \to \{0,1\}$ where each g_i is either a constant 0 or 1, or one of the input variables $x_1, \dots, x_n$, or is obtained from the previous functions by applying an Or, And or Not operation. The length t of this sequence is the *size* of the circuit. Given a multi-output function $f : \{0,1\}^n \to \{0,1\}^k$, we may consider it as the family of boolean (i.e., single-output) functions $f_1, \dots, f_k$ such that, for every input $a \in \{0,1\}^n$, $f_i(a)$ is the i-th bit of the output $f(a)$. A circuit computes such a function f if all the functions $f_1, \dots, f_k$ belong to it.

The main difficulty in proving non-trivial lower bounds on the size of circuits is the presence of Not gates. The effect of such gates on circuit size remains to a large extent a mystery. It is, therefore, worth to recall what we actually know about this "ghost." Among the basic questions concerning the role of Not gates are the following:

- Given a function f, what is the minimum number $M(f)$ of Not gates in a circuit computing f?
- Given a circuit, to what extend can we decrease the number of Not gates in it without a substantial increase in its size?
- Suppose that a function f in n variables *can* be computed by a circuit of size polynomial in n, but every circuit with $M(f)$ negations computing f requires superpolynomial size. What is then the minimal number $R(f)$ of negations sufficient to compute f in polynomial size? In other words, how many Not gates do we need in oder to achieve superpolynomial savings in circuit size?

In this lecture we give some (partial) answers to these questions.

6.1. Markov's theorem

More than 30 years ago, Markov (1957) has made an intriguing observation that *every* Boolean function on n variables can be computed by a circuit with only

$\lceil \log(n+1) \rceil$ negations.[1] In fact, he announced a stronger result. A *multivalued* function on n variables is just a function $f : \{0,1\}^n \to \{0,1\}^k$. We can consider such a (mult-output) function as the sequence $f = (f_1, \ldots, f_k)$ of Boolean (i.e., single-output) functions such that, for every input $a \in \{0,1\}^n$, $f_i(a)$ is the i-th bit of the output $f(a)$. If $k = 1$, then the function is *boolean*. For two binary vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ we write $x \leq y$ if $x_i \leq y_i$ for all i. We also write $x < y$ if $x \leq y$ and $x_i < y_i$ for at least one i. A function f is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$.

A *chain* in the binary n-cube is an increasing sequence $y^1 < y^2 < \cdots < y^k$ of vectors in $\{0,1\}^n$. The *decrease* $d_Y(f)$ of a function $f = (f_1, \ldots, f_k)$ on a chain Y is the number of indices i such that $f(y^i) \not\leq f(y^{i+1})$. [2] The *decrease* $d(f)$ of f is the maximum of $d_Y(f)$ over all chains Y. In [**20**] Markov announced the following surprisingly tight result.

Theorem 6.1. (Markov 1957) *For every function f, the minimum number of Not gates contained in a circuit computing f is precisely $M(f) := \lceil \log(d(f)+1) \rceil$.*

Proof. Let $\mathrm{neg}(f)$ denote the minimum number of negation gates contained in a circuit computing f. Our goal is to prove that, for every function f,

$$\mathrm{neg}(f) = \lceil \log(d(f)+1) \rceil.$$

For simplicity, we consider only boolean functions f (the case of multi-output functions is similar and we leave it as an exercise). We first prove the **lower bound**:

$$\mathrm{neg}(f) \geq \lceil \log(d(f)+1) \rceil.$$

Fix a chain $Y = \{y^1 < y^2 < \cdots < y^k\}$ for which $d_Y(f) = d(f)$, and let $I(f) = \{i : f(y^i) > f(y^{i+1})\}$ be the set of the (indices of) *inverses* made by function f on this chain; hence, $|I(f)| = d_Y(f) = d(f)$. Take an arbitrary circuit C computing f, and let r be the number of negation gates in it. Our goal is to show that $r \geq \lceil \log(|I(f)| + 1 \rceil$.

To show this, let g be the function computed on the output of the first negation gate of C. Since the function computed at the input of this gate is monotone, $d_Y(g) \leq 1$, i.e., g can make at most one inverse on Y. If $d_Y(g) = 0$ then $g \equiv 0$ or $g \equiv 1$, and we can replace this negation gate by a constant. Otherwise, there is an i_0 such that $g(y^i) = 1$ for all $i \in I_1 = \{1, \ldots, i_0\}$, and $g(y^i) = 0$ for all $i \in I_0 = \{i_0+1, \ldots, k\}$. Depending on whether $|I_1 \cap I(f)| \geq |I(f)|/2$ or not, replace the gate g by constant 1 or by constant 0. In both cases the resulting circuit has one negation gate fewer, and computes a function f_1 for which $d_Y(f_1) \geq |I(f)|/2 = d(f)/2$.

Now, if the original circuit C would have $r \leq \lceil \log(|I(f)| + 1 \rceil - 1$ negations, then repeating this argument r times we would obtain a circuit without any negations which computes a function f_r for which $d_Y(f_r) \geq 1$. But the function f_r is monotone, a contradiction.

Let us now prove[3] the **upper bound**:

$$\mathrm{neg}(f) \leq \lceil \log(d(f)+1) \rceil.$$

---

[1] Here and in what follows, all logarithms are base two; hence, $\lceil \log(n+1) \rceil$ is the number of bits in the binary representation of n.

[2] That is, the number of indices i such that for some j, $f_j(y^i) > f_j(y^{i+1})$.

[3] We follow the argument due to Fischer (1996).

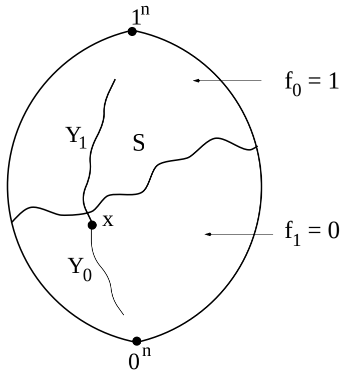

Figure 1. Chain Y_0 ends in x, and Y_1 starts with x.

We will prove the upper bound by induction on

$$M(f) := \lceil \log(d(f) + 1) \rceil.$$

Base: $M(f) = 0$. Then $d(f) = 0$, so f is monotone and $\operatorname{neg}(f) = 0$.

Induction Step: Suppose $\operatorname{neg}(f') \leq M(f')$ for all boolean functions f' such that $M(f') \leq M(f) - 1$.

Let S be the set of all vectors $x \in \{0,1\}^n$ such that for every chain Y *starting* with x we have

$$d_Y(f) < 2^{M(f)-1}. \tag{1}$$

We can show that for every chain Y *ending* in a vector outside the set S we also have

$$d_Y(f) < 2^{M(f)-1}. \tag{2}$$

To show this, assume that there is a chain Y_0 ending in a vector $x \notin S$ and such that $d_{Y_0}(f) \geq 2^{M(f)-1}$ (see Fig. 1). The fact that x does not belong to S means that there must be a chain Y_1 starting in x for which $d_{Y_1}(f) \geq 2^{M(f)-1}$. But then the decrease $d_{Y_0 \cup Y_1}(f)$ of f on the combined chain $Y_0 \cup Y_1$ is

$$d_{Y_0 \cup Y_1}(f) = d_{Y_0}(f) + d_{Y_1}(f) \geq 2^{M(f)} = 2^{\lceil \log(d(f)+1) \rceil} > d(f),$$

a contradiction with the definition of $d(f)$.

Consider now two functions f_0 and f_1 defined as follows:

$$f_1(x) = \begin{cases} f(x) & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

and

$$f_0(x) = \begin{cases} 1 & \text{if } x \in S \\ f(x) & \text{if } x \notin S \end{cases}$$

By (1) and (2) we have $d(f_i) \leq 2^{M(f)-1} - 1$ for both $i = 0, 1$. Hence,

$$M(f_i) = \lceil \log(d(f_i) + 1) \rceil \leq \lceil \log 2^{M(f)-1} \rceil = M(f) - 1.$$

By the induction hypothesis, $\mathrm{neg}(f_i) \le M(f_i) \le M(f) - 1$ for both $i = 0, 1$. It therefore remains to show that

$$\mathrm{neg}(f) \le 1 + \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}. \tag{3}$$

For this we need one auxiliary result. Let $\mu(y, y', x)$ be a boolean function in $n+2$ variables $y, y', x_1, \ldots, x_n$. Say that μ is a *connector* of two boolean functions $f_0(x)$ and $f_1(x)$ if for $i = 0, 1$

$$\mu(i, \neg i, x) = f_i(x),$$

that is, if $\mu(0, 1, x) = f_0(x)$ and $\mu(1, 0, x) = f_1(x)$.

Claim 6.2. *Every pair of functions* $f_0(x)$, $f_1(x)$ *has a connector* μ *such that* $\mathrm{neg}(\mu) \le \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}$.

Proof of Claim. We argue by induction on $r = \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}$. If $r = 0$ then both functions f_i are monotone, and we can take $\mu(y, y', x) = (y \wedge f_1) \vee (y' \wedge f_0)$. *Induction Step.* Let $C_i(x)$ be a circuit with $\mathrm{neg}(f_i)$ negations computing $f_i(x)$. Replacing the first Not gate in C_i by a new variable z we obtain a circuit $C_i'(z, x)$ on $n+1$ variables which contains one Not gate fewer. Let $f_i'(z, x)$ be the function computed by this circuit; hence, $\mathrm{neg}(f_i') \le r - 1$. Moreover, if $h_i(x)$ is the monotone function computed immediately before the first Not gate (in C_i) then

$$f_i(x) = f_i'(\neg h_i(x), x).$$

By the induction hypothesis, there is a boolean function $\mu'(y, y', z, x)$ (the connector of the pair f_0', f_1') such that $\mathrm{neg}(\mu') \le \max\{\mathrm{neg}(f_0'), \mathrm{neg}(f_1')\} \le r - 1$ and for $i = 0, 1$

$$\mu'(i, \neg i, z, x) = f_i'(z, x).$$

It is easy to see that then the function $\mu(y, y', x)$ obtained from $\mu'(y, y', z, x)$ by replacing the variable z by the function[4]

$$Z(y, y', x) := \neg\big((y \wedge h_1(x)) \vee (y' \wedge h_0(x))\big)$$

is a connector of f_0 and f_1. Since the functions h_0 and h_1 are monotone, we have[5] $\mathrm{neg}(\mu) \le 1 + \mathrm{neg}(\mu') \le r$, as desired. □

Now we can finish the proof of the theorem as follows. Let $s(x)$ be the characteristic function of S. Note that $s(x)$ is monotone (show this!). Let μ be a connector of f_0 and f_1. Then

$$f(x) = \mu\big(s(x), \neg s(x), x\big)$$

and by Claim,

$$\mathrm{neg}(f) \le 1 + \mathrm{neg}(\mu) \le 1 + \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}.$$

This completes the proof of (3), and thus, the proof of the theorem. □

Ex. 6.3. Show that the maximum of $M(f)$ over all functions on n variables is:
(i) $\lceil \log(n+1) \rceil$ in the case of multi-output functions, and
(ii) $\lfloor \log(n+1) \rfloor$ in the case of single-output (i.e., boolean) functions.

---

[4] $Z(0, 1, x) = \neg h_0(x)$ and $Z(1, 0, x) = \neg h_1(x)$

[5] In the first inequality we use the following simple fact: If $f = g(h_1(x), \ldots, h_m(x))$ then $\mathrm{neg}(f) \le \mathrm{neg}(g) + \max_i \mathrm{neg}(h_i)$.

Ex. 6.4. Let $f : \{0,1\}^n \to \{0,1\}^k$ be a k-output function, and $f_1, \dots, f_k$ be the corresponding family of boolean functions. Show that $d(f) \leq \sum_{i=1}^{k} d(f_i)$.

6.2. Fischer's theorem

According to Markov's theorem, every function in n variables *can* be computed by a circuit with at most

$$M(n) := \lceil \log(n+1) \rceil$$

Not gates. Fischer (1974) has shown that restricting the number of negations to $M(n)$ entails only a polynomial blowup in circuit size!

Theorem 6.5. (Fischer 1974)[6] *If a function on n variables can be computed by a circuit of size t, then it can be computed by a circuit of size at most $2t + O(n^2 \log^2 n)$ using at most $M(n)$ Not gates.*

Proof. It is easy to show (do this!) that every circuit of size t can be transformed to a circuit of size at most $2t$ such that all negations are placed only on the input variables. Hence, it is enough to show how to compute the (multi-output) function

$$NEG(x_1, \dots, x_n) = (\neg x_1, \dots, \neg x_n)$$

by a circuit of size $O(n^2 \log^2 n)$ using $M(n)$ negations.

A simple (but crucial) observation is that[7]

$$\neg x_i = f_i(x)$$

where[8]

$$f_i(x) := \bigwedge_{k=0}^{n} \left(\neg T_k^n(x) \vee T_{k,i}^n(x) \right)$$

and

$$T_{k,i}^n(x_1, \dots, x_n) := T_k^{n-1}(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n).$$

It can be shown (we will not do this) that all the functions T_k^n and $T_{k,i}^n$ $(0 \leq k \leq n, 1 \leq i \leq n)$ can be computed by a *monotone* circuit of size $O(n^2 \log^2 n)$. It remains to compute the function $\neg T(x) := \left(\neg T_1^n(x), \neg T_2^n(x), \dots, \neg T_n^n(x) \right)$ using at most $\lceil \log(n+1) \rceil$ negations. To do this, we first take a monotone circuit $C_1(x)$ computing the function $T(x) := \left(T_1^n(x), T_2^n(x), \dots, T_n^n(x) \right)$. Observe that the outputs of this circuit belong to the set A_{sort} of all inputs $y \in \{0,1\}^n$ whose bits are *sorted* in decreasing oder $y_1 \geq y_2 \geq \dots \geq y_n$. Using only $\lceil \log(n+1) \rceil$ negations it is possible to construct a circuit $C_2(y)$ of size $O(n)$ which computes $NEG(y)$ correctly on all inputs in A_{sort} (see Exercise 6.6). The resulting circuit $C(x) = C_2(C_1(x))$ computes $\neg T(x)$. □

Ex. 6.6. Let $n = 2^r - 1$. Construct a circuit C_n of size $O(n)$ which has at most r Not gates such that $C_n(y) = NEG(y)$ for all inputs $y \in A_{sort}$. *Hint*: Let $y = (y_1, \dots, y_n) \in A_{sort}$. Take the middle bit y_m $(m = n/2)$ and show that the output of C_n can be obtained from the output of $C_{n/2}$ and the output of

---

[6] In fact, the term $O(n^2 \log^2 n)$ here can be replaced by $O(n \log n)$.

[7] Let $a \in \{0,1\}^n$. If $\neg x_i(a) = 1$ then $a_i = 0$, implying that in this case $T_k^n(a) = T_{k,i}^n(a)$ for all $k = 1, \dots, n$, and hence, $f_i(a) = 1$. If $\neg x_i(a) = 0$ then $a_i = 1$. So, if we take $k := |a|$ (= the number of 1's in a) then $T_k^n(a) = 1$ and $T_{k,i}^n(a) = 0$, implying that $f_i(a) = 0$.

[8] Here, as before, $T_k^n(x_1, \dots, x_n)$ is the k-th threshold function: $T_k^n(x) = 1$ iff x has at least k 1's.

$\neg y_m$. For this observe that $\neg y_m = 1$ implies $\neg y_j = \neg y_m$ for all $j > m$, whereas $\neg y_m = 0$ implies $\neg y_j = \neg y_m$ for all $j < m$.

Ex. 6.7. Fischer's theorem holds for every function f in n variables, but requires $\lceil \log(n+1) \rceil$ negations. Suppose that f outputs 1 on all inputs with at least k 1's, and outputs 0 on all inputs with at most l 1's. Show that then Fischer's simulation requires only $\lceil \log(k-l) \rceil$ negations.

6.3. Can Fischer's simulation be improved?

Markov's theorem motivates the following question.

> Suppose that a function f in n variables *can* be computed by a circuit of size polynomial in n (in this case we say that f is *feasible*), but every circuit with $M(f)$ negations computing f requires superpolynomial size. What is then the minimal number $R(f)$ of negations sufficient to compute f in polynomial size?

Fischer's result only implies that this number lies somewhere between the Markov's lower bound $M(f)$ and the generic upper bound $\lceil \log(n+1) \rceil$. The discrepancy between these two bounds is especially evident for functions with $M(f) = 0$, i.e., for monotone functions. Berkowitz [**7**] and Valiant [**33**] have shown that for so-called *slice* functions (these are monotone boolean functions which are non-trivial only on one slice of the n-cube) negations are powerless, i.e., cannot lead to a superpolynomial savings.[9] So, the first natural question was whether using Not gates can lead to superpolynomial savings in size *at all*?

This (long standing) problem was resolved by Razborov [**26**]: there exist explicit feasible monotone boolean functions f such that $R(f) > 0$. The function here is the characteristic function of bipartite graphs containing a perfect matching. Another example of such a function (requiring even larger monotone circuits) was given by Tardos [**32**]. The next natural question was: *how large does $R(f)$ actually is*? Or, in other words:

> How many Not gates do we actually need to achieve superpolynomial savings in circuit size?

This last question was considered by many authors under additional restrictions on the topology and/or on the use of Not gates. In particular, Okolnishnikova [**22**], and Ajtai and Gurevich [**4**] have shown that there exists monotone functions that can be computed with polynomial size, *constant* depth circuits, but can not be computed with *monotone*, polynomial size, constant depth circuits. This implies that $R(f) > 0$ in the class of AC^0 circuits. Moreover, it was shown by Satha and Wilson [**29**] that in this class we need much more than $\lceil \log(n+1) \rceil$ negations: there is a (multi-output) function computable in constant depth that cannot be computed in constant depth with $o(n/\log^{1+\varepsilon} n)$ negations. (Note that this result does not contradict with the Markov–Fischer upper bound: their simulation requires logarithmic depth.) Another line of research was to restrict the use of Not gates. For circuits of logarithmic depth, a lower bound $R(f) = \Omega(n)$ was proved by Raz and Wigderson [**25**] under the restriction that *all* the negations are placed on the input variables: there is an explicit monotone function (corresponding to the connectivity problem for graphs) that can be computed with polynomial size, depth $O(\log^2 n)$

[9] See Exercises 10.12 and 10.13 i the book.

circuits, but can not be computed with polynomial size, depth $k \log n$ circuits using only $o(n/2^k)$ negated variables.

However, if we put restrictions neither on the depth nor on the use of Not gates, then the only we knew so far was that some feasible monotone functions f_n require $R(f_n) > 0$ negations. It was even not clear[10] whether there are feasible monotone functions for which $R(f_n) \to \infty$ as $n \to \infty$.

In the next section we show that, somewhat surprisingly, the results of Razborov and Tardos can be used to move the threshold $R(f)$ (of the first superpolynomial decrease in size) much nearer to the Markov–Fischer bound $\lceil \log(n+1) \rceil$: there are (explicit) feasible monotone functions $f : \{0,1\}^n \to \{0,1\}^n$ such that $R(f) \geq \log n - c \log\log n$.

6.4. Feasible functions requiring many Not gates

The idea is very simple, and is based on the following observation. Let $C = C(Y_0, Y_1)$ be a circuit which computes two Boolean functions $f_0 = f_0(Y_0)$ and $f_1 = f_1(Y_1)$ on disjoint sets of variables, $Y_0 \cap Y_1 = \emptyset$. A *minterm* of a monotone boolean function is a minimal set of variables which, if assigned the value 1, forces the function to take the value 1 regardless of the values assigned to the remaining variables.

Observation 6.8. *If C has only one Not gate, then at least one of the two functions f_0, f_1 can be computed by a* monotone *circuit of the same (or smaller) size.*

Proof. Let g be the monotone(!) Boolean function computed at the input to the Not gate (i.e., immediately before this gate). Since g is monotone, all its minterms are positive (no negated literals). Hence, we have only two possibilities: either some minterm of g lies entirely in Y_1, or not. In the first case we assign constant 1 to all the variables in Y_1, whereas in the second case we assign constant 0 to all the variables in Y_0. As the function g is monotone, in both cases it turns into a constant function (constant 1 in the first case, and constant 0 in the second), and the subsequent Not gate can be eliminated. Since $Y_0 \cap Y_1 = \emptyset$, the setting $Y_\varepsilon \mapsto \varepsilon$ does not affect the function $f_{1-\varepsilon}$. Hence, we obtain a circuit which computes either f_0 or f_1 and contains no Not gates, i.e., is monotone. □

Observation 6.8 gets rid of only one Not gate. However, it can be directly extended to the case of many such gates by taking more copies of boolean functions.

Let $f = f(X)$ be a Boolean function in m variables $X = \{x_1, \ldots, x_m\}$, and $n = km$. A function $f_n : \{0,1\}^n \to \{0,1\}^k$ is a *k-fold extension* of f if it computes k copies of f on disjoint copies $X_1, \ldots, X_k$ of X. That is, given an input $(a^1, \ldots, a^k)$ with $a^i \in \{0,1\}^{X_i}$, the function outputs the sequence $\big(f(a^1), \ldots, f(a^k)\big)$. Important here is that the i-th output bit $f(a^i)$ is independent of inputs a^j for $j \neq i$. Note also that the extension of monotone functions are monotone.

Iterating the argument used in the proof of Fact 6.8 immediately yields the following

[10] A step in this direction was made by Amano and Maruoka [?] who proved that the clique function cannot be computed in polynomial size using only $(1/6) \log\log n$ negations. Unfortunately, this does not answer the question, because the clique function itself is not feasible (unless $P = NP$).

Observation 6.9. *Let f be a monotone Boolean function, and k be a nonnegative integer. If f cannot be computed by a monotone circuit of size t, then its k-fold extension cannot be computed by a circuit of size t using $\lceil \log(k+1) \rceil$ negations.*

As a corollary we obtain

Theorem 6.10. ([**16**]) *There exist explicit feasible monotone functions $f_n : \{0,1\}^n \to \{0,1\}^n$ such that $R(f_n) \geq \log n - 9 \log\log n$.*

Proof. Take an explicit monotone Boolean function T_m in m variables considered in [**32**]. As shown in [**32**], this function is feasible (can be computed by a non-monotone circuit of size $m^{O(1)}$) but every monotone circuit computing it requires size is exponential in $\Omega(m^{1/8})$. Let $n = km$ where $k = 2^r$ and

$$r = \left\lceil \log \frac{n}{(\log n)^9} \right\rceil$$

and consider the k-fold extension f_n of T_m. Then f_n can be computed by a (non-monotone) circuit of size at most $k \cdot m^{O(1)}$, which is, of course, polynomial in n. On the other hand, by Fact 7.9, every circuit with at most r Not gates computing f_n must have size exponential in $\Omega(m^{1/8}) = \Omega\left((2^{-r}n)^{1/8}\right) = \Omega\left((\log n)^{9/8}\right)$. Thus, $R(f_n) > r \geq \log n - 9 \log\log n$. □

One may ask what happens if besides Not gates we will allow, say, any monotone *real-valued* functions $\varphi : \mathsf{R}^2 \to \mathsf{R}$ as gates—does then the use of negations can still lead to a drastic decrease of size? The question makes sense because it is known[11] (see [**?**]) that there exist a lot of monotone boolean functions (slice functions) that can be computed by monotone real circuits (no Not gates) of linear size $O(n)$, but require boolean circuits with And, Or and Not gates of size $2^{\Omega(n)}$. On the other hand, it is shown in [**15**] that Tardos' function T_m requires also monotone real circuits of size $2^{\Omega(m^{1/8})}$. This means that the function f_n considered in Theorem 6.10 captures the role of negations in a quite strong sense: this function is feasible but cannot be computed by a circuit of polynomial size with fewer than $\log n - 9 \log\log n$ Not gates, even if we allow *all* monotone real-valued functions as gates.

The message of this theorem is that, in the context of the P vs. NP problem, it is important to understand the role of Not gates when their number r is *indeed* very close to the Markov–Fisher upper bound of $\lceil \log(n+1) \rceil$: moving by more than $\log\log n$ apart from this bound we essentially move by $\log n$ and reach the world of monotone circuits ($r = 0$) where Not gates play no role—they don't exist in this world at all!

> **Ex. 6.11.** (*Research problem*) Unlike Tardos' function T_m, the function f_n in Theorem 6.10 has many output bits. Find an explicit monotone *boolean* (i.e., single-output) function f_n for which $R(f_n) = \Omega(\log n)$.

When approaching this problem the following observation may be (apparently) useful.

Observation 6.12. *Let f be a boolean function and suppose that it can be computed by a circuit of size t with at most r negations. Then for any $A \subseteq f^{-1}(0)$ and $B \subseteq f^{-1}(1)$, there is a monotone boolean function g such that g can be computed*

[11] See Exercises 10.14 and 10.15 in the book.

by a monotone circuit of size at most t and either g or its negation $\overline{g}$ rejects a 2^{-r} fraction of inputs from A and accepts a 2^{-r} fraction of inputs from B.

Proof. We argue by induction on r. If $r = 0$, then we choose the circuit itself. If $r \geq 1$, then consider the first negation gate and the function g that is computed at the gate immediately before this negation gate. Let $\varepsilon \in \{0, 1\}$ be such that $g(a) = \varepsilon$ for at least one half of the inputs $a \in A$. If also one half of the inputs $b \in B$ have $g(b) = \varepsilon \oplus 1$, then either g or $\overline{g}$ has the property stated in the lemma. Otherwise (if $g(b) = \varepsilon$ for one half of the inputs $b \in B$) we replace the negation gate by the constant $\varepsilon \oplus 1$ and use the induction hypothesis for the sets $A_1 \subseteq A$ and $B_1 \subseteq B$ of inputs on which g outputs the constant ε. □

Now if a boolean function f can be computed by a circuit with small number of negations then, according to Observation 7.1, there must be a *monotone* boolean function g such that either g or its negation $\overline{g}$ separates some two large enough subsets of inputs $A_0 \subseteq f^{-1}(0)$ and $A_1 \subseteq f^{-1}(1)$. If this pair is separated by the function g itself, then we can try to argue in a standard way to show that g cannot be t-simple if t is too small, and then apply the lower bounds criterion (Theorem 10.16 in the book). The problem, however, is when the pair A_0, A_1 is separated by $\overline{g}$. Then g also separates this pair, but does this in the "wrong direction:" $g(A_0) = 1$ and $g(A_1) = 0$. This is a problem because for the original function to be not t-simple the 0/1-terms of f must be scattered enough, meaning that they must be short (much shorter than $n/2$); hence, their complements must be long, implying that the corresponding terms of g may be no more scattered well enough.