# Lecture 5. Formula complexity I

A Boolean function can be computed by different models, and an important one is the class of formulas. Roughly speaking, formulas are circuits with fanout 1. To be more specific, a formula is a rooted tree s.t.

* each internal node is associated with a gate which is either AND or OR,
* each leaf is associated with a variable xi, the negation of a variable , or a constant 0 or 1.

(Note that we have used DeMorgan rule to push all negations to the leaf level.) The formula computes a function f on input x by evaluating the gates in a bottom-up manner, and the root outputs f(x).

Different measures of formula are studied, among which depth, size (number of gates) and leaf size (number of leaves) are the most important ones. The leaf size L(F) of a formula F is simply the number of leaves. The leaf size L(f) of a function f is the minimum leaf size L(F) of any formula F that computes f.

A function is *monotone* if whenever . In other words, it requires that changing an input variable from 0 to 1 doesn’t decrease the function value. For monotone functions, we can require the formula contain only variables and constants in leaves, but no negation of variables. (Convince yourself.) These are called *monotone formulas*. So for monotone functions, we can define the *monotone leaf size* as the minimal number of leaves of any monotone formula that computes f.

While circuit lower bounds are too hard to prove---the best unconditional lower bound is only cn for some constant c, people stepped back and tried to prove formula lower bounds. In this and the next lectures, we will learn various methods to prove lower bounds for formula complexity.

## (Random) Restriction

The idea of restriction is as follows. We will pick some variables and fix their values to be some constants, resulting in a “subfunction” f’. The variables are picked and fixed in such a ways that the complexity of this subfunction is considerably smaller than that of f. But if we keep doing so, we’ll go to a very small scale function with complexity 0. But this small scale function is still not a constant function, thus has a positive complexity---contradiction. This idea went all the way back to Subbotovskaya [Sub61].

**Lemma 1.1**. For every Boolean function f of n variables, it is possible to fix one variable such that the resulting subfunction satisfies

*.*

Before proving the lemma, let’s get some feelings of the parameters and see why it’s nontrivial. Suppose F is a minimal formula which computes f, and F has s = L(f) leaves. Since f has n variables, there exists one variable xi which appears at least s/n times in the leaf level (though some of the appearances are in the form of ). If we fix *xi* to be some constant, then the leaf size will reduce to . The lemma says that actually one can reduce the size by more. Now we prove it.

*Proof*. Define F and s as in the above discussion. The idea is that if there is a subtree H in F which is of the form for some subtree G, then setting will kill the whole subtree G. (You can handle the cases where the gate is OR, or where the occurrence of xi is in negation, in a similar way.) Therefore, suppose variable i appears times, then for each appearance there is an assignment of xi = 0/1 to kill its “neighbor” subtree G. Then one of the two assignments, xi = 0 and xi = 1, kills more, at least t/2, subtrees. Use that assignment. The leaf size reduces to at most

,

as claimed.

The above analysis is almost correct except that it forgets to consider the case in which the subtrees G intersect. Consider the extreme case that there is a subtree H of the form This looks weird since clearly some xi’s are redundant. We can indeed make this rigorous by claiming the following: *For any leaf or , its neighbor subtree G doesn’t contain variable i*. (Thus the G’s don’t intersect.)

The claim is not hard to prove. Basically, if we see and G contains xi, then it is observed that one can simply replace the xi in G by 1 without changing the function. (If xi = 1, then we “guessed correctly” by replacing xi in G by 1. If xi = 0, then anyway.) Other cases such as G contains , or the gate connecting and G is OR, can be handled similarly. This completes the proof of the lemma. □

Repeatedly applying the lemma gives the following theorem.

**Theorem 1.2**. For every Boolean function f of n variables and every , there is a way to fix variables such that the resulting subfunction of k variables satisfies

*.*

Proof. Applying the lemma times, we obtain a subfunction of k variables with leaf size at most

*.* □

In the above theorems, we carefully constructed an assignment to shrink the leaf size. Next we will show that actually this construction doesn’t need to be so careful---a random one does the job with good probability.

**Theorem 1.3**. Let f be a Boolean function of n variables. Pick k variables uniformly at random, and for the rest variables, assign each of them a random constant . Then with probability at least 3/4,

.

Proof. Let’s choose a variable randomly and assign a random value. (We will choose variables one by one; note that the process is actually the same as the random assignment in the statement of the theorem.) Denote by s1 the number of resulting leaves. The proof of the previous theorem actually shows that the expected number of reduced leaves is at least . Thus

Continue the above process for variables, resulting in a subfunction of k variables with the expected leaf size

.

The claimed statement now follows from Markov Inequality. □

*(The above analysis is a bit informal about the expectation of multiple stages. Can you make it more formal? Just compute the stage 2, namely E[s2], and you’ll see why it’s correct.)*

We can apply the theorem to show a lower bound for the parity function .

**Theorem 1.4**. For the parity function, .

Proof. Fix all but one variable, resulting in the parity function (or its negation) of one variable, which has leaf size at least 1. So

,

which implies the claimed lower bound. □

## Rectangle and tiling number

We now introduce another lower bound method for leaf size, due to Khrapchenko [Khr72] and Rychkov [Ryc85].

A *rectangle* is a pair (A,B) of disjoint sets . A rectangle (A,B) is *monochromatic* if there exists an , s.t. for all and *,* it holds that . A rectangle (A,B) is *1-monochromatic* if there exists an , s.t. for all and , it holds that and . A rectangle (A,B) is *0-monochromatic* if there exists an , s.t. for all and , it holds that and . Any monochromatic rectangle is either 1- or 0-monochromatic. (See why?)

The *tiling number* of a rectangle R is the minimum number r s.t. R can be partitioned into r monochromatic rectangles. Please note that by “partition”, we emphasize that the rectangles are disjoint (and, of course, their union needs to be the whole set R). The *tiling number* of a function f is , the tiling number of the rectangle .

**Theorem 2.1**. For every Boolean function f, . If f is monotone, then .

*Proof*. We prove the first inequality by induction on L(f), and the second one is similar. The base is easy to confirm: If , then or . Then the rectangle is itself monochromatic by definition.

Now the induction step. Take a formula F with the minimal leaf size. Assume that the root is AND function; the other case can be handled similarly. So the function , and . (Convince yourself that the subtrees are also optimal for the corresponding subfunctions.) Define

, and .

First, we observe that

The first two inclusions are because imples that . (Recall that .) The third one is by definition of B0. The fourth one is because . The last one is trivial by definition of B0 and B1. Now

 // def. of

 // tiling separately

 // the four inclusions above

 // def. of

 // induction hypothesis

 □

Now we need to further lower bound .

**Theorem 2.2**. Let , , and . We have

where is the Hamming weight of .

*Proof*. Suppose we have a decomposition of into r monochromatic rectangles Ri of dimensions . Let *ci* be the number of elements of S in Ri. We claim that each . Indeed, suppose that all elements (x,y) in Ri differ at position *j*: . Then each x has at most one y s.t. , therefore Similarly, we can show that Then

Rearranging the terms gives the bound. □

We can use this to show a quadratic lower bound for parity function, improving the bound given in the last section.

**Theorem 2.3**. Let f be the parity function of n variables. Then

*Proof*. Clearly, , and . So □

One may wonder whether this lower bound can be further improved. The answer is no: actually there is a formula to compute parity function with leaves. Can you find one?

(*hint: Consider the two-variable version, , first.*)

## References

[Sub61] B. A. Subbotovskaya, **Realizations of linear functions by formulas using +, ., -**, *Soviet Math. Dokl.* 2, 110–112, 1961.

[Khr72] V.M. Khrapchenko, **A method of obtaining lower bounds for the complexity of -schemes**, *Math. Notes Acad. of Sci. USSR* 10, 474–479, 1972.

[Ryc85] K. L. Rychkov, **A modification of Khrapchenko’s method and its application to lower bounds for -schemes of code functions**, in *Metody Diskretnogo Analiza*, 42 (Novosibirsk), 91–98, 1985.