CSC2429 / MAT1304: Circuit Complexity — January 10, 2019

Lecture 1: DeMorgan Circuits and Formulas

Instructor: Benjamin Rossman — *Scribe: Lily Li*

Overview

Section 1: **Administrivia.**

Section 2: **Basic Definitions.** Define Boolean function f, DeMorgan circuits, circuits size $\mathcal{C}(f)$, leaf size $\mathcal{L}(f)$, and depth of a circuit.

Section 3: **Uniform vs. Concrete Models of Computation.** Define uniform and non-uniform models of computation. Circuits can efficiently simulate Turing Machines.

Section 4: **Balancing Formulas.** Spira 1971, relates leaf-size of a formula and its depth.

Section 5: **Circuit Size of Almost All Boolean Functions.** Lupanov 1958, upper bound of $O(2^n/n)$ for all n-ary Boolean functions. Shannon 1949, proved matching lower bound for almost all functions.

1 Administrivia

- **Instructor:** Ben Rossman.
- **Course Info:** Available at the course website. Just in case the website is down: lectures are Thursdays from 16:00 to 18:00 in Bahen B026. Office hours are by appointment.
- **Textbook:** *Boolean Function Complexity* by Stasys Junkha. This is available as a free eBook through the University of Toronto library.
- **Prerequisites:** No formal course prerequisites. Please read Appendix A.1 of the textbook (background on probability) and make sure you understand the material.
- **Workload:** Homework assignment(s), scribe notes, short paper report (max 10 pages), and optional class presentation.

2 Basic Definitions

Definition 1. *A n-**ary Boolean function** f is a function of the form $f : \{0,1\}^n \to \{0,1\}$. Usually we interpret $(0,1)$ as $(\mathsf{FALSE}, \mathsf{TRUE})$ or as $(1,-1)$ — this makes sense if you think of it as $(-1)^0$ and $(-1)^1$.*

Let $\{0,1\}^* = \cup_{n\in\mathbb{N}}\{0,1\}^n$. We typically refer to a (family of) Boolean function(s) $f : \{0,1\}^* \to \{0,1\}$. This corresponds to a sequence of functions $f_n : \{0,1\}^n \to \{0,1\}$ and to a language $L \subseteq \{0,1\}^*$ described by its characteristic function $f_L : \{0,1\}^* \to \{0,1\}$.

Example 2. *The following are some examples of n-ary Boolean functions:*

1. $PARITY(x_1, ..., x_n) = \sum_{i=1}^{n} x_i \mod 2$.
2. $MOD_p(x_1, ..., x_n) = 1 \iff \sum_{i=1}^{n} x_i \equiv 0 \mod p$.
3. $MAJORITY_n(x_1, ..., x_n) = 1 \iff \sum_{i=1}^{n} x_i \geq \lceil n/2 \rceil$.
4. $k - CLIQUE : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$. *Think of each graph G as an indicator vector $\mathbb{1}_G$ over its $\binom{n}{2}$ edges. Then $k - CLIQUE(\mathbb{1}_G) = 1$ if and only if G has a k-clique.*

Let us consider DeMorgan circuits. These contain logical connectives $\{\vee, \wedge, \neg\}$, input variables $\{x_1, ..., x_n\}$, and constants $\{0,1\}$.

Definition 3. *A n-**ary DeMorgan circuit** is a finite directed acyclic graph (DAG) with nodes labelled as follows:*

- *Nodes of in-degree zero ("inputs") are labelled by a variable or a constant.*
- *Non-input nodes ("gates") of in-degree one are labelled with $\neg$. Gates of in-degree two are labelled with $\vee$ or $\wedge$.*
- *A subset of the nodes are designated as "outputs" (default assumption: there is a single output node with out-degree zero).*

*An n-ary circuit with m output gates computes a function $\{0,1\}^n \to \{0,1\}^m$. Two circuits are **equivalent** if they compute the same function.*

Formulas are tree-like circuits. Since different branches in a formula depend on different copies of the variables, formulas are memory-less. See Figure 1. Proving that formulas are polynomially weaker than circuits is still an open problem.

Definition 4. *The **size** of a circuit is the number of $\vee$ and $\wedge$ gates it contains.*

*The **leaf-size** of a formula is the number of leaves in its associated DAG. This is one more than the circuit size as defined above.*

*The **circuit size** of an n-ary Boolean function $f : \{0,1\}^n \to \{0,1\}$, written $\mathcal{C}(f)$, is the minimum size of a circuit computing f. Similarly, the **formula (leaf) size** of f, written $\mathcal{L}(f)$, is the minimum size of a formula computing f.*

*The **depth** of a circuit is the maximum number of $\wedge$ and $\vee$ gates on any input to output path.*

For example, note that $\mathcal{C}(\text{AND}_n) = n-1$ and $\mathcal{L}(\text{AND}_n) = n$. The upper bounds are obvious and the lower bounds follow from the fact that any circuit of size $n-1$ or formula of leaf-size n depends on at most n variables (easy exercise). In the next lecture, we will show that $\mathcal{C}(\text{PARITY}_n) = 3(n-1)$ and $\mathcal{L}(\text{PARITY}) = \Theta(n^2)$.

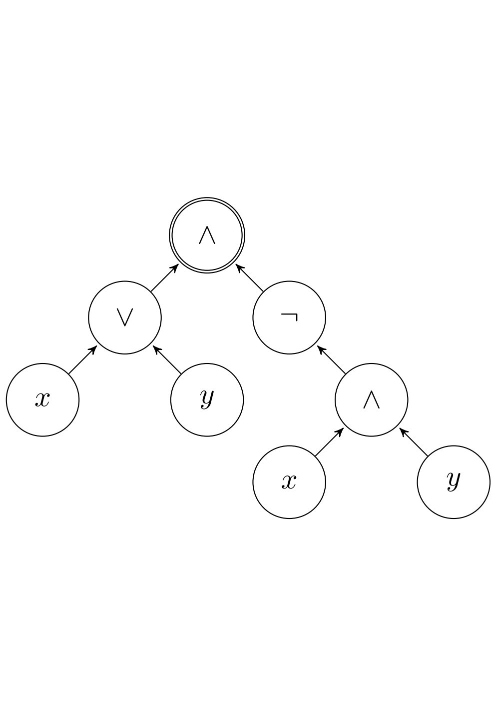

Figure 1: (Left) Formula computing $x \oplus y$. (Right) Circuit computing $x \oplus y \oplus z$.

2.1 Other ways of measuring size

Other ways of counting the size of a circuit include: (1) counting the number of wires and (2) counting all gate types (including $\neg$ gates). It turns out that the result of these calculations differ from our definition above by at most a factor of two.

It is often convenient to consider circuits and formulas where negations appear only above variables.

Definition 5. *A circuit is in* ***negation normal form*** *if the input to every $\neg$ gate is a variable. See Figure 2.*

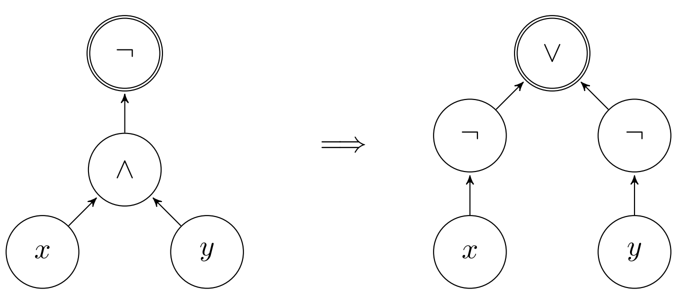

Figure 2: Apply DeMorgan's Law to all $\neg$ gates whose inputs are not literals on the left circuit to get the equivalent right circuit in negation normal form.

Every circuit can be put into negation normal form by repeatedly applying DeMorgan's Law to push all $\neg$ gates to the bottom. Once a circuit is in negation normal form, we can forget about $\neg$

gates if we instead interpret inputs as literals (variables or their negations).

Claim 6. *Every formula F is equivalent to a formula of the same leaf-size in negation normal form. Every circuit C of size m is equivalent to a circuit in negation normal form of size $\leq 2m$.*

The factor 2 blow-up for circuits comes from the fact that for each gate g in C, we need gates representing the functions g and $1 - g$ (in the case where g has outgoing wires to both a $\neg$ gate and a $\wedge$ or $\vee$ gate).

2.2 General Bases (sets of gates)

A **basis** B is a set of Boolean functions (or "gate types"). Examples of basis include:

- DeMorgan basis: $\{\wedge, \vee, \neg\}$.
- Full binary basis: all Boolean functions $\{0,1\}^2 \to \{0,1\}$ (for example, you would get $\oplus$).
- Monotone basis: $\{\wedge, \vee\}$ (NOT universal).
- AC^0 basis: $\{\wedge^k, \vee^k, \neg : k \in \mathbb{N}\}$ which are unbounded fan-in $\wedge$ and $\vee$ gates.

For a function f, let $\mathcal{L}_B(f)$ and $\mathcal{C}_B(f)$ be the leaf and circuit size of f with formulas and circuits built from gates of basis B. A basis is **universal** if it computes all functions. For any two universal bases B_1 and B_2 it is possible to build a circuit using gates from B_1 which simulates any gate from B_2. If all functions in B_1 and B_2 have constant arity, it follows that $\mathcal{C}_{B_2}(f) = O(\mathcal{C}_{B_1}(f))$; for formula size, the relation is $\mathcal{L}_{B_2}(f) = \mathcal{L}_{B_1}(f)^{O(1)}$. (Exercise: Make sure you understand why.) This polynomial blow-up is unavoidable in some cases. Recall the function $PARITY_n$: in the DeMorgan basis $\mathcal{L}_{\{\wedge,\vee,\neg\}}(PARITY_n) = \Theta(n^2)$, whereas $\mathcal{L}_{\text{full binary basis}}(PARITY_n) = \mathcal{L}_{\{\oplus\}}(PARITY_n) = n - 1$.

3 Uniform vs. Concrete Models of Computation

Definition 7. *A* ***uniform model of computation*** *is a single machine/program with a finite description which operates on all inputs in $\{0,1\}^*$. Examples range from simple finite automata (where we have lower bounds ala the pumping lemma) to complex Turing Machines (lower bounds much harder to come by).*

Recall that a language $L \subseteq \{0,1\}^*$ can be interpreted as a sequence of functions $(f_0, f_1, ...)$ where $f_n : \{0,1\}^n \to \{0,1\}$ and $f_n(\mathbf{x}) = 1 \iff \mathbf{x} \in L$ for any $\mathbf{x} \in \{0,1\}^n$. A ***non-uniform (concrete) model of computation*** *is a sequence $(C_0, C_1, ...)$ of combinatorial objects (namely circuits) where C_n computes f_n. Examples include: circuits in the DeMorgan basis, restricted class of circuits (formulas, monotone model), decision trees, etc.*

Observe that the non-uniform model of computation is more powerful than the uniform one since the finite program can be used as every combinatorial objects in the sequence. It follows that lower bounds in the non-uniform model also imply lower bounds in the uniform model. While upper bounds in the uniform model imply upper bounds in the non-uniform model. *We want: unconditional lower bounds.*

Circuits efficiently simulate Turing Machines.

Lemma 8. *Any Turing Machine (TM) M with running time $t(n)$ can be simulated by a circuit (family of) of size $O\left(t(n)^2\right)$.*

Exercise for the reader. *Hint: think about configurations of the Turing Machines as a $t(n)\times t(n)$ grid and construct a circuit for every grid cell.* Fischer and Pipenger (1979) proved an $O(t(n)\log t(n))$ upper bound on *oblivious Turing Machines*[1]. It is unknown if we can do better.

Corollary 9. *If there is a super polynomial lower-bound (better than $\Omega(n^c)$ for all constants $c > 0$) on the circuit size of any language in NP, then $\mathsf{P} \neq \mathsf{NP}$.*

Finding the lower-bound would actually show $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ where $\mathsf{P/poly}$ is the class of languages decidable by poly(n)-size circuits.

We will see some polynomial lower bounds for formulas in the DeMorgan basis later on. As a historical curio, the following is a catalogue of lower bound results for an explicit Boolean function:

1. $\Omega(n^{1.5})$ Subboboskay '61
2. $\Omega(n^2)$ Khrapchenko '71
3. $\Omega(n^{2.5-o(1)})$ Andreev '83
4. $\Omega(n^{3-o(1)})$ Håstad '98 (this is the state of the art until very recently).

4 Balancing Formulas

Next we consider the relationship between depth and formula leaf-size (in the DeMorgan basis). First observe that every circuit of depth d is equivalent to a formula of depth d and size at most 2^d. To see this, take the circuit and duplicate any branches that get reused. The resulting binary tree has at most as many nodes as a perfect binary tree of depth d, namely 2^d.

The next theorem shows the converse: that every formula of size s can be "balanced" to obtain a formula of depth $O(\log s)$.

Theorem 10. *(Spira 1971). Every formula with leaf-size s is equivalent to a formula of depth $O(\log s)$ ($2\log_{3/2}(s)$ to be exact) and thus size at most $s^{O(1)}$ ($s^{2/\log_2(3/2)}$).*

Proof. By induction on s. The base case is trivial. Let F be the original formula and g be some gate. Let F_g be the sub-formula rooted at g. For $b \in \{0,1\}$, let $F^{(g\leftarrow b)}$ be the formula with g replaced with the constant value b. See Figure 3. Note that $\mathcal{L}(F) = \mathcal{L}\left(F_g\right) + \mathcal{L}\left(F^{(g\leftarrow b)}\right)$. Minimize $\mathcal{L}(F)$ by balancing the two terms on the RHS. By Claim 11, we can find a gate g such that $\frac{s}{3} \leq \mathcal{L}(F_g) \leq \frac{2s}{3}$.

[1]An oblivious TM is one whose head motion depends only on the size of the input and not its particular bits.

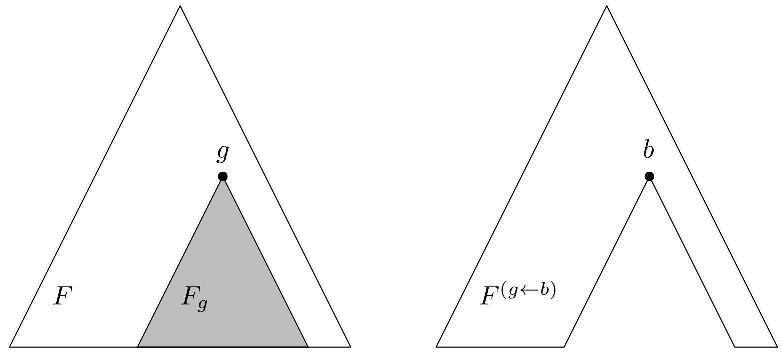

Figure 3: Illustration of gate g and formulas F_g and $F^{g\leftarrow b}$.

Note that $F \equiv \left(F_g \wedge F^{(g\leftarrow 1)}\right) \vee \left((\neg F_g) \wedge F^{(g\leftarrow 0)}\right)$; F_g must evaluate to 0 or 1 and the formula does just that. Apply the induction hypothesis to the four formulas F_g, $F^{(g\leftarrow 1)}$, $\neg F_g$, and $F^{(g\leftarrow 0)}$ to get formulas of depth $\leq 2\log_{3/2}(2s/3)$. The original formula F can only grow by at most depth two so

$$\begin{aligned}
\text{depth}(F) &\leq \max\left\{\text{depth}\left(F_g\right), \text{depth}\left(F^{(g\leftarrow 1)}\right), \text{depth}\left(\neg F_g\right), \text{depth}\left(F^{(g\leftarrow 0)}\right)\right\} + 2 \\
&\leq 2\log_{3/2}\frac{2s}{3} + 2 \\
&= (2\log_{3/2} s - 2) + 2 \\
&= 2\log_{3/2} s
\end{aligned}$$

Thus there exists a formula equivalent to F of depth at most $O(\log s)$. $\square$

Claim 11. *There exists a gate g such that F_g has leaf-size between $\frac{s}{3}$ and $\frac{2s}{3}$ leaves.*

Proof. Let $r \rightsquigarrow \ell$ be a root to leaf path in the DAG containing the most $\wedge/\vee$ gates. At the root r, $\mathcal{L}(F_r) = \mathcal{L}(F) = s$ and at the leaf ℓ, $\mathcal{L}(F_\ell) = 1$. Starting at r and moving down to ℓ, the successive leaf-sizes can at most halve after each step. Thus there must exists a gate g for which $\frac{s}{3} \leq \mathcal{L}(F_g) \leq \frac{2s}{3}$. $\square$

5 Circuit Size of Almost All Boolean Functions

5.1 Upper Bound

Given a function $f : \{0,1\}^n \to \{0,1\}$, let us consider some upper bounds for $\mathcal{C}(f)$.

1. Brute force DNF: $O(n2^n)$. There are 2^n rows in the truth table of f. Each row specifies the output given the n inputs. Thus a clause with $n-1$ $\wedge$ gates represents each row of the table. Formally we consider the expression

$$f(\mathbf{x}) = \bigvee_{\mathbf{a}\in f^{-1}(1)} (\mathbf{x} = \mathbf{a}) = \bigvee_{\mathbf{a}\in f^{-1}(1)} (l_1 \wedge l_2 \wedge \cdots \wedge l_{n-1} \wedge l_n)$$

where $l_i = x_i$ if $a_i = 1$ and $l_i = \overline{x}_i$ otherwise.

2. Function decomposition: $O(2^n)$. Observe that

$$f(\mathbf{x}) \equiv (x_n \wedge f_1(\mathbf{x})) \vee (\overline{x}_n \wedge f_0(\mathbf{x}))\,.$$

where $f_1 = f(x_1, ..., x_{n-1}, 1)$ and $f_0 = f(x_1, ..., x_{n-1}, 0)$. Thus

$$\mathcal{C}(f) \leq \mathcal{C}(f_1) + \mathcal{C}(f_0) + 3.$$

Apply the decomposition recursively to f_1 and f_0. Generally at step k,

$$\mathcal{C}(f) \leq \sum_{\mathbf{a} \in \{0,1\}^k} \mathcal{C}(f_{\mathbf{a}}) + 3(2^k - 1)$$

where $f_{\mathbf{a}}(\mathbf{x}) = f(x_1, ..., x_{n-k}, a_1, ..., a_k)$. Since $f(\mathbf{a})$ is a constant at the n^{th} step, $3(2^n - 1)$ is an upper bound on the circuit size of f.

3. Computation reuse: $O(2^n/n)$. See Theorem 13 below.

Let $ALL_n : \{0,1\}^n \to \{0,1\}^{2^{2^n}}$ be the function which calculates all the n-ary Boolean functions at the same time[2]. That is $(ALL_n(\mathbf{x}))_f \coloneqq f(\mathbf{x})$ for any n-ary Boolean function f.

Claim 12. $\mathcal{C}(ALL_n) \leq O(2^{2^n})$.

Proof. Similar to the function decomposition analysis. For every function f in the output of ALL_n, $f(\mathbf{x}) \equiv (x_n \wedge f_1(\mathbf{x})) \vee (\overline{x}_n \wedge f_0(\mathbf{x}))$ where $f_1 = f(x_1, ..., x_{n-1}, 1)$ and $f_0 = f(x_1, ..., x_{n-1}, 0)$. Note that f_1 and f_0 are outputs of ALL_{n-1}. See Figure 4. Since ALL_n has 2^{2^n} outputs,

$$\mathcal{C}(ALL_n) \leq \mathcal{C}(ALL_{n-1}) + 3\left(2^{2^n}\right) = c\left(2^{2^{n-1}}\right) + 3\left(2^{2^n}\right) \in O\left(2^{2^n}\right)$$

for some constant c. ☐

Theorem 13. *(Lupanov 1958) Every n-ary Boolean function has circuit size $O(2^n/n)$*

Proof. The key idea is to use ALL_{n-k} in place of $\{f_{\mathbf{a}} : \mathbf{a} \in \{0,1\}^k\}$ in the analysis of function decomposition. Formally, we have

$$\mathcal{C}(f) \leq \sum_{\mathbf{a} \in \{0,1\}^k} \mathcal{C}(f_{\mathbf{a}}) + 3(2^k - 1) \leq \mathcal{C}(ALL_{n-k}) + 3(2^k - 1) \leq O\left(2^{2^{n-k}}\right) + O(2^k)$$

where the last inequality follows from Claim 12. Observe that the two terms on the RHS are balanced when $k = n - \log(n - \log n)$ since

$$\begin{aligned} O\left(2^{2^{n-k}}\right) + O\left(2^k\right) &= O\left(2^{2^{\log(n-\log n)}}\right) + O\left(2^{n-\log(n-\log n)}\right) \\ &= O\left(2^{n-\log n}\right) \\ &= O\left(2^n/n\right) \end{aligned}$$

It follows that the circuit complexity of all n-ary Boolean function is bounded above by $O(2^n/n)$. ☐

[2] To see that the range of ALL_n is indeed 2^{2^n}, recall that the domain of every n-ary Boolean function is 2^n. There is a bijection between the set of functions and the power set of $\{0,1\}^n$ (of size 2^{2^n}).

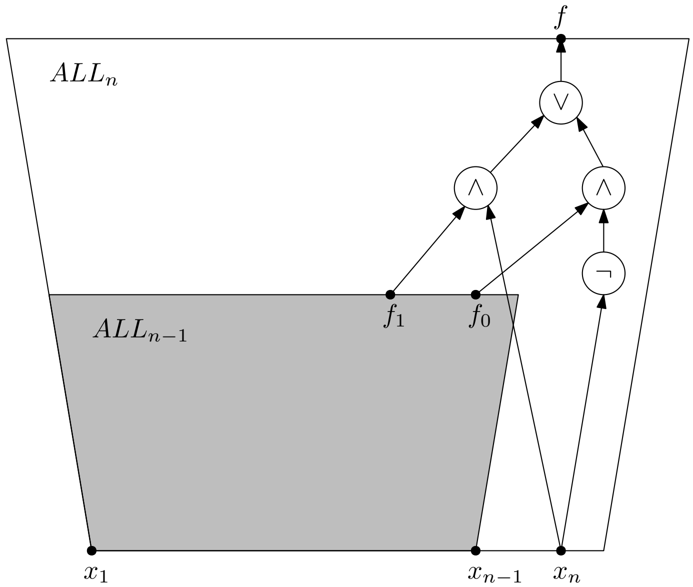

Figure 4: Obtaining a circuit for ALL_n from a circuit for ALL_{n-1}.

5.2 Lower Bound

Prior to Lupanov's result above, Shannon showed a matching lower bound.

Theorem 14. *(Shannon 1949) Almost all n-ary Boolean functions (as $n \to \infty$) have circuit size $O(2^n/n)$.*

Proof. Use a counting argument. Recall the number of n-ary Boolean functions is 2^{2^n} and let $s = \frac{2^n}{n}$. We will show that the number of Boolean functions which can be computed by circuits of size s is $\ll 2^{2^n}$. Let A be the set of all n-ary circuits with $2n$ literals, $x_1, ..., x_n, \overline{x}_1, ..., \overline{x}_n$, and s gates, denoted $g_1, ..., g_s$. We obtain an upper bound on the number of circuits in A as follows. Each circuit can use any subset of the s gates. Each $\wedge/\vee$ gate can pick two inputs from the $2n$ literals and $s-1$ other gates. If n is sufficiently large (say $n \geq 100$), then $s + 2n < 3s$ so

$$|A| \leq 2^s (s+2n)^{2s} \leq 18^s s^{2s}.$$

Observe that every n-ary function with $\mathcal{C}(f) \leq s$ is computed by at least $s!$ distinct circuits in A, since we can permute the labels on the s gates. (Exercise: Complete this observation by showing that f is computed by some $C \in A$ which has no symmetries under permutations of gate labels $g_1, \ldots, g_s$.) Thus the total number of Boolean functions computed by circuits in A is at most $\frac{|A|}{s!}$. Recall that $s! \geq \left(\frac{s}{e}\right)^s$. For $s = \frac{2^n}{n}$,

$$\frac{|A|}{s!} \leq \frac{18^s s^{2s}}{(s/e)^s} \leq 50^s s^s = 50^{2^n/n} \left(\frac{2^n}{n}\right)^{2^n/n} = \left(\frac{50}{n}\right)^{2^n/n} (2^n)^{2^n/n} \leq 2^{2^n - 2^n/n}$$

since $n \geq 100$. Thus at least 2^s Boolean formulas have circuit size greater than s. $\square$