CSE 531: Computational Complexity I **Winter 2016**

Lecture 2: Circuit complexity vs Turing machine complexity

January 8, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

1 Circuits

A *Boolean circuit* C is a directed acyclic graph with $n+2$ source nodes labelled by n input bits $x_1, \ldots, x_n$ and two constants 0 and 1 for some $n \in \mathbb{N}$. Non-source nodes are called *gates* and will have fan-in (indegree) 1 or 2 (though we will sometimes relax this bound). Fan-in 2 gates are labelled $\wedge$ or $\vee$ and fan-in 1 gates are labelled $\neg$. (This is called the De Morgan basis.) There is a single sink node (the output node).

Each node computes a function of x in the following obvious way. The value of each input is the corresponding bit value. For a gate f labeled $\wedge$ with predecessor gates g and h, value $f(x)$ is $g(x)\mathrm{AND}h(x)$. Similarly, for gates labeled $\vee$ using OR and gates labelled $\neg$ is NOT. The function computed by C, $C(x)$ is the function computed by the output gate.

The *size* of C is the # of non-source vertices of C. The *depth* of C is the length of the longest path in C from an input to the output.

Definition 1.1. *Define* $\mathbb{B}_n = \{f : \{0,1\}^n \to \{0,1\}\}$.

We can consider more general circuits than De Morgan circuits that have binary gates computing arbitrary functions in $\mathbb{B}_2$, which is called the *complete* basis. It is easy to see that each element α of $\mathbb{B}_2$ can be computed by a circuit of constant size and depth and by substituting this circuit for each node labelled by α. This changes size and depth by at most a constant factor.

Example 1.2.

Definition 1.3. *A circuit C is* satisfiable *iff there is some input y such that $C(y) = 1$.*

Definition 1.4. *A family of circuits $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ computes a function $f : \{0,1\}^* \to \{0,1\}$ iff $C_{|x|}(x) = f(x)$ for all $x \in \{0,1\}^*$. The* size *of $\mathcal{C}$ is $S(n)$ iff $|C_n| \leq S(n)$ for all $n \in \mathbb{N}$.*
The depth *of $\mathcal{C}$ is $d(n)$ iff the depth of C_n is $\leq d(n)$ for all $n \in \mathbb{N}$.*

Note that the definition does not require any relationship between the C_n for different values of n and so a circuit family may not in general have a finite description. This is the essence of non-uniformity.

In particular, circuit families of small size may compute undecidable functions such as the function f that outputs 1 on input x if and only if $x = 1^n$ for some n such that the n-th Turing machine M_n halts on input n. For each n the circuit C_n for this function will either always output 0 or it will be a tree of $\wedge$ gates checking that the input is 1^n. This family has circuit size n. (One can even define a family of size 1 that also computes an undecidable function by changing f to g which outputs 1 on all inputs x with $|x| = n$ where the n-th TM M_n halts on input n. For each n , the n-th circuit computes either the constant function 0 or the constant function 1.)

Define $\mathsf{SIZE}(\mathsf{S}(\mathsf{n})) = \{\mathsf{f} : \{0,1\}^* \to \{0,1\} \mid \mathsf{f}$is computed by a circuit family $\mathcal{C}$ of size $S(n)\}$.
$\mathsf{DEPTH}(\mathsf{d}(\mathsf{n})) = \{\mathsf{f} : \{0,1\}^* \to \{0,1\} \mid \mathsf{f}$is computed by a circuit family $\mathcal{C}$ of depth $d(n)\}$.

Define $\mathsf{P/poly} = \bigcup_{\mathsf{k}} \mathsf{SIZE}(\mathsf{n}^{\mathsf{k}})$. (The notation POLYSIZE is sometimes used for this class; we will motivate the P/poly notation later.)

Special classes of circuits A circuit ϕ is a *formula* iff each gate has fan-out (outdegree) at most 1; i.e., the circuits is a directed tree. For every circuit there is an equivalent formula of the same depth but potentially much larger size since one needs to replicate subcircuits of the circuit whenever the outdegree is larger than 1. Note that changing the basis may change formula size by much more than a constant factor. (In particular using a basis that includes $\oplus$ (parity) lets one compute the parity of n bits in size $O(n)$, whereas in the De Morgan basis it requires size $\Theta(n^2)$.)

A formula is a *DNF (Disjunctive Normal Form)* formula iff all $\neg$ gates are next to input nodes, and after that, each path from an input or $\neg$ gate to the output consists of $\wedge$ gates followed by $\vee$ gates. We can reexpress this equivalently in terms of two levels of unbounded fan-in gates with an unbounded $\vee$ gate at the output that receives fan-in from unbounded $\wedge$ gates that each receive as input variables or their negations (which are both called *literals*. The unbounded fan-in $\wedge$ gates are called *terms*.

CNF (Conjunctive Normal Form) formulas are defined dually, with an unbounded fan-in $\wedge$ gate at the output, receiving inputs from unbounded fan-in $\vee$ gates. The $\vee$ gates are called *clauses*.

A CNF formula is a k-CNF iff all of its clauses have fan-in k. Similarly a k-DNF is a DNF formula with all terms having fan-in k.

By including one term of size n for each input assignment in on which f outputs 1 (or, dually, one clause of size n for each assignment on which f output 0) we immediately obtain the following proposition.

Proposition 1.5. *Every function $f \in \mathbb{B}_n$ has DNF formulas and CNF formulas of size $O(n2^n)$.*

You will show on homework that you can do somewhat better than this using circuits, namely $O(2^n/n)$.

Relationships between circuits and Turing machines

Theorem 1.6 (Pippenger-Fischer)**.** *For every time-constructible $T(n)$,*

$$\mathsf{DTIME}(\mathsf{T}(\mathsf{n})) \subseteq \bigcup_{\mathsf{c}} \mathsf{SIZE}(\mathsf{cT}(\mathsf{n}) \log_2 \mathsf{T}(\mathsf{n})).$$

This immediately implies the following relationship.

Corollary 1.7. $\mathsf{P} \subset \mathsf{P/poly}$.

Proof. The proof of Theorem 1.6 follows very closely along the key ideas of Cook's proof of the Cook-Levin Theorem as modified by Ladner. The *configuration* (also called a snapshot in the text) of a Turing machine consists of

- the contents of the nonblank region of all its tapes
- the position of all its read heads
- its current state

We first describe the basic idea assuming that the time $T(n)$ machine M is a 1-tape Turing machine. In this case we can use an option mark on each cell to indicate the current state if the head is scanning that cell.

Since there are at most $T(n)$ non-blank cells on each tape, there is a sequence of configurations $C_0, C_1, \ldots, C_{T(n)}$ each of length $O(T(n))$ that can be written out in a $T(n) \times T(n)$ tableau, with each C_t as a row and each column representating the state of a fixed cell on the tape at each of the time steps. The simulation will allocate a constant number of gates with each cell in the tableau. Each cell then contains an element from $\Gamma \times (Q \cup \square)$.

The circuit for the initial configuration C_0 is immediately constructible using the basic definitions. In order to determine the contents of cell i at time t, it suffices to know the contents of cells $i-1$, i, and $i+1$ at time $t-1$ and the transition function of the Turing machine being simulated. This can be computed using a simple circuit of constant size. Each entry in the tableau will have an essentially identical bit of circuitry that connects these 4 cells together. (The left boundary will be slightly different because only 2 cells from the previous layer are relevant.)

This circuit now correctly computes the contents of the tape cells and it suffices to check to see whether the state at level $T(n)$ is q_{accept}. This is essentially a full binary fan-in tree of $\wedge$ gates. The total size is $O(T^2(n))$. Note that since one can convert a TM to a single-tape TM by only squaring the size, this already suffices to prove that $\mathsf{P} \subset \mathsf{P/poly}$.

It will be to important (for use in the proof of the Cook-Levin theorem) to note that given $[M]$ and a bound $T(n)$, it is very easy to construct this simulating tableau circuit of size $O(T^2(n))$.

In order to obtain the better $O(T(n)\log T(n))$ size circuit, we will use the efficient oblivious universal Turing machine U instead of M. The key advantage to being oblivious is that we only need to build in the circuitry for contents that might change during each time step. Since U is oblivious, there is only one tableau entry per row in which the head can lie. This allows us to eliminate most of the cells in the tableau. and only retain the cells immediately before or after that are of interest. This only a constant number of cells per time step are represented which is size $O(T(n)\log T(n))$.

The one difficulty this creates is that how the contents to be written into the current cell are determined depends, potentially, on cells that were last written at vastly different prior times. Here is where we use the fact discussed in the construction of U that the head positions are not only oblivious, but also very simply predictible, which given the t and i allows the algorithm to pick out exactly which prior time-step was the last to modify a cell since only those entries of the tableau will be simulated by the circuitry. □

2 Nondeterminism and NP

Definition 2.1. NP *is the set of languages $L \subseteq \{0,1\}^*$ such that there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM V (called a* verifier*) such that for all $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} V(x,y) = 1.$$

Such a y is called a certificate/witness/proof *that $x \in L$.*

Since we can choose $p(n) = 0$ for all $n \in \mathbb{N}$, we immediately have

Proposition 2.2. $\mathsf{P} \subseteq \mathsf{NP}$.

Some NP problems

- $CIRCUIT\text{-}SAT = \{[C] \mid C \text{ is a satisfiable Boolean circuit}\}$
- $SAT = \{[\phi] \mid \phi \text{ is a satisfiable Boolean formula}\}$
- $k\text{-}SAT = \{[\phi] \mid \phi \text{ is a satisfiable } k\text{-CNF formula}\}$.

In each of these cases the certificate y is the assignment that makes the circuit/formula evaluate to 1. The verifier V propagates the assignment through the circuit and outputs the value produced by the circuit. This is clearly polynomial-time computable. The problem that this verifier solves is an important problem in its own right.
$CIRCUIT - VALUE = \{([C], y) \mid C(y) = 1\}$ and the argument shows that $CIRCUIT\text{-}VALUE \in \mathsf{P}$.