Lecture 5

Circuit Complexity / Counting Problems

April 12, 2004
Lecturer: Paul Beame
Notes: William Pentney

5.1 Circuit Complexity and Uniform Complexity

We will conclude our look at the basic relationship between circuit complexity classes and uniform complexity classes. First, we will prove that circuits can efficiently solve problems in uniform complexity classes.

Theorem 5.1 (Pippenger, Fischer). *If $T(n) \geq n$, then $\mathsf{TIME}(T(n)) \subseteq \bigcup \mathsf{SIZE}(kT(n)\log_2 T(n))$.*

First, consider the following variant of the traditional k-tape TM:

Definition 5.1. A multitape TM is *oblivious* if the motion of each head depends only upon the length of its input.

Note that an oblivious TM's motion has nothing to do with the input itself; given an input of size n, the machine's head(s) will perform the *same* series of motions regardless of the original contents of the tape. We will prove two lemmas from which Theorem 5.1 follows immediately.

The first is a more careful version of the original result of Hennie and Stearns showing that k-tape TMs can be efficiently simulated by 2-tape TMs. The key extension is that the resulting TM can be made oblivious.

Lemma 5.2 (Hennie-Stearns, Pippenger-Fischer). *For $T(n) \geq n$, if $A \in \mathsf{TIME}(T(n))$ then A is recognized by a 2-tape deterministic oblivious TM in time $O(T(n)\log T(n))$.*

The second lemma shows that oblivious TMs can be efficiently simulated by circuits.

Lemma 5.3. *[Pippenger-Fischer] If $A \subseteq \{0,1\}^*$ is recognized by a 2-tape deterministic oblivious TM in time $O(T(n))$, then $A \in \bigcup_k \mathsf{SIZE}(kT(n))$ (i.e. size linear in $T(n)$.*

First we will prove Lemma 5.3 whose proof motivates the notion of oblivious TMs.

Proof of Lemma 5.3. Consider the tableau generated in the standard proof of the Cook-Levin theorem for an input of size n for a Turing machine running in $O(T(n)$. The tableau yields a circuit of size $O(T^2(n))$, with "windows" of size 2×3. However, on any given input the TM head will only be in one cell at a given time step; i.e., one cell per row in the tableau. With a normal TM, though, it is not possible to anticipate which cell that will be.

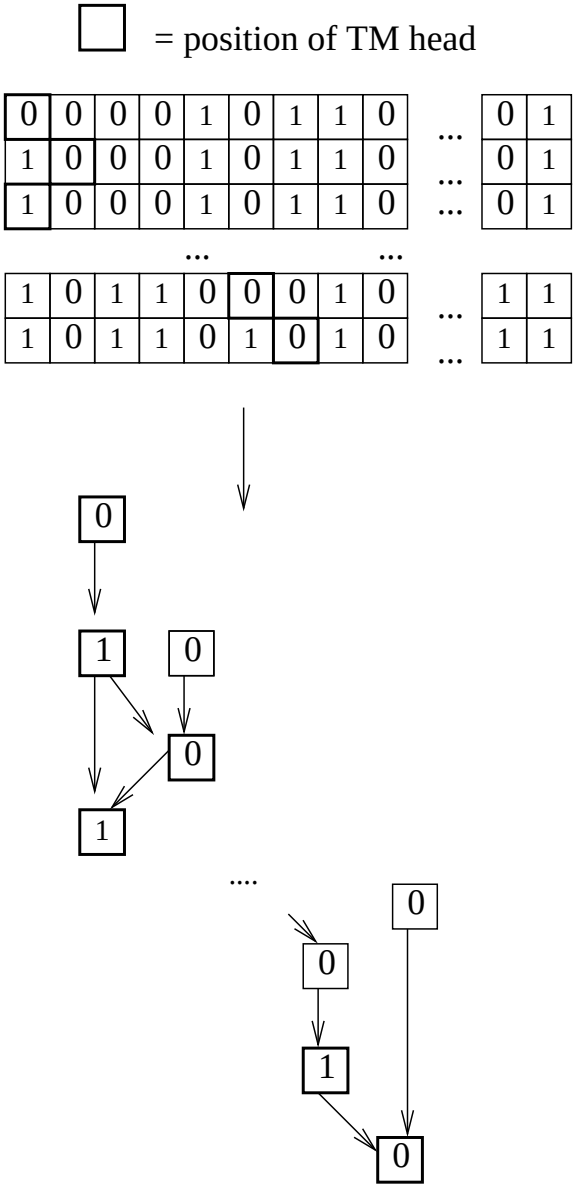

Figure 5.1: Representing an oblivious TM using constant-size windows

.

Since our TM M is oblivious, we know the path that the head shall take, and where it will be at each timestep, simply by knowing the length of the input. We know if and when a given cell on the tape will be revisited, and can ignore it at other times. Thus we can use "windows" containing simply the contents of a cell from the last time it was active and the contents of the previous cell visited to see what the head moves, to represent each step taken by that head, and all other cells in the tableau will be irrelevant to that step in the computation. This is represented in Figure 5.1. The subsequent configuration resulting from any configuration can thus be computed by a circuit of constant size. Since the tableau is of height $T(n)$, we can express the tableau in size $O(T(n))$. □

Now, we prove Lemma 5.2, that if $A \in \mathsf{TIME}(T(n))$, then A is recognized by a 2-tape deterministic

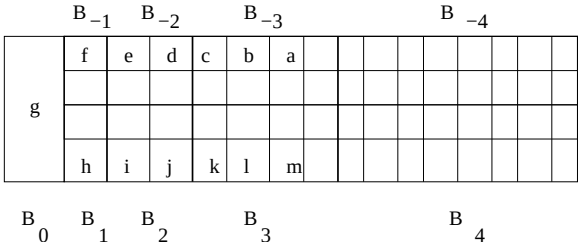

Figure 5.2: 4-track oblivious simulation of a TM tape.

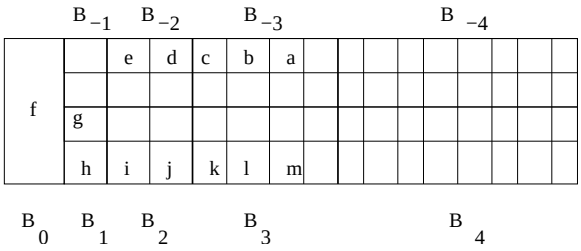

Figure 5.3: The simulation in Figure 5.2 when the head moves left. Block B_1 is bunched and B_{-1} stretched.

and oblivious TM in time $O(T(n) \log T(n))$.

Proof of Lemma 5.2. We will build a 2-tape TM M' which will consist of two tapes: one holding all the contents of M's tapes, and one "work tape".

Traditionally, we think of a TM as having a tape which the head moves back on forth on. For this proof, it may help to instead think of the TM head as being in a fixed position and each of the tapes as moving instead, similar to a typewriter carriage (if anyone remembers this ancient beasts!). Of course a tape would be an infinitely long typewriter carriage which would be too much to move in a single step. Instead, we think of moving the tapes back and forth, but we may not do so smoothly; sometimes we will be "bunching up" a tape, such that some cells may be effectively layered on top of others and other parts will be "stretched", parts that are not bunched or stretched are "smooth".

It will be convenient to concentrate on simulating one tape of M at a time. In reality we will allocate several tracks on tape 1 of M' to each of the k-tapes. It is simplest also to imagine that each of M's tapes is 2-way infinite, although M' will have only 1-way infinite tapes.

More precisely, imagine the scenario modeled in Figure 5.2. For each tape of M, we have four tracks – two for those cells to the left of the current cell, and two for those to the right of the current cell. The "outer" tracks will contain the contents of the tape when that portion of the tape is smooth; the "inner" tracks will contain portions of the tape that are bunched up. The current cell is always on the left end of the set of tracks. The tracks wrap around as the head of M moves.

We will divide the tracks into "blocks", where each block will consist of a portion of the two upper or lower tracks. Block B_0 contains the current cell (at the left end of the tape); block B_1 is the lower half of the the cell immediately to the right of block B_0 and has capacity for up to 2 cells of M; block B_2 is the lower half of the next 2 cells to the right of that with capacity for up to 4 cells of M, etc. In general, block B_i has contains the lower half of the next 2^{i-1} cells starting at cell 2^{i-1} and has capacity for 2^i cells of M. Block B_{-i} for $i \geq 1$ contains the corresponding upper half of cells of M' whose lower half is the block B_i. In general, for block B_i and B_j, if $i < j$ then the contents of block B_i will be to the left of those for block

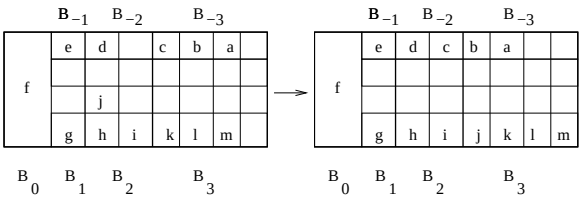

Figure 5.4: The smoothing of blocks B_2 and block B_{-2}.

B_j. The outer tracks of block B_i or B_{-i} will contain contents of cells of M that are closer to the cell in block B_0 than the inner tracks.

In our simulation, when we wrap the tracks around, we permit the tape to "bunch up" within a block; when this happens, the contents of the cell of M that would ordinarily be in the outer track of the next block is now placed in the next free cell on the inner track of the current block. We can later undo this by moving the bunched-up data in block B_i into B_{i+1}; this process is called *smoothing* the block. Similarly, a block B_i may be stretched, so that it contains fewer than 2^{i-1} cells from the tape; if other data is pushed into it, the vacant area on the track is filled. The tape is smoothed when it is no longer stretched. Figure 5.3 shows the movement of the tracks in our simulation depicted in Figure 5.2 when the head moves left; block B_1 is bunched and block B_{-1} stretched. In Figure 5.4, we see the smoothing of blocks B_2 and B_{-2} on such a simulation.

We maintain the following invariants after simulating each time step t of the simulation:

1. Every block B_i with $|i| > \lceil \log_2 t \rceil + 1$ is smooth.
2. $B_i + B_{-i} = 2^{|i|}$ i.e. if B_i is bunched, B_{-i} is stretched an equal amount.
3. Each B_i consists of consecutive elements of the tape of M.
4. If t is a multiple of 2^i then blocks $B_i, B_{i-1}, B_{i-2}..., B_1, B_0, B_{-1}, B_{-2}, ...B_{-i}$ are smooth.

To maintain our invariants, whenever t is $2^i t'$ for some odd t', we smooth out blocks $B_{-i}...B_i$ by borrowing from or adding entries to B_{i+1} and $B_{-(i+1)}$. We do this by copying the contents of all the cells in $B_1, \ldots, B_i, B_{i+1}$ onto tape 2 and then writing them back into those cells only on the outer tracks, except, possibly, for block B_{i+1}. The same is done for blocks $B_{-1}, \ldots, B_{-(i+1)}$. We need to argue that this smoothing can be accomplished.

After smoothing blocks $B_{-(i+1)}, \ldots, B_{(i+1)}$ when t is a multiple of 2^{i+1}, we may simulate M for up $2^i - 1$ steps without accessing blocks B_{i+1} or $B_{-(i+1)}$. On the 2^i-th step the number of entries that will need to appear in blocks $B_0, B_1, \ldots, B_i$ (or $B_{-i}, B_{i-1}, \ldots, B_0$) is between 0 and 2^{i+1} since this number can have changed by at most 2^i during this time. Since B_{i+1} has remained smooth during this time, there are precisely 2^i occupied cells in B_{i+1} and space for 2^i more inputs. Therefore there enough entries in B_{i+1} to borrow to fill the up to 2^i spaces in blocks $B_0, \ldots, B_i$ that need to be filled, or enough space to take the overflow from those blocks. A similar argument applies to blocks $B_{-i}, \ldots, B_0$. Therefore this smoothing can be accomplished. By carefully making sure that the inner and outer tracks are scanned whether or not they contain entries, the smoothing can be done with oblivious motion.

How much does the smoothing cost when t is a multiple of 2^i but not 2^{i+1}? It will be less than or equal to $c2^i$ steps in tota for some constant c. We will perform smoothing $\frac{1}{2^i}$ of the time, with cost $c2^i$ per smoothing. Therefore, the total time will be $\sum_{i \leq \lceil \log_2 T(n) \rceil + 1} c2^i \frac{T(n)}{2^i}$ which is $O(T(n) \log T(n)$. □

We can use the proof of THeoremm 5.1 to prove the following theorem:

Theorem 5.4 (Cook 83). *If $T(n) \leq n$ and $L \in \mathsf{NTIME}(T(n))$ then there exists a reduction that maps input x to a 3-CNF formula ϕ_x in $O(T(n)\log T(n))$ time and $O(\log T(n))$ space such that ϕ_x has size $O(T(n)\log T(n))$ and $x \in L \Leftrightarrow \phi_x \in 3SAT$.*

Proof. ϕ_x is usual 3-CNF based on CIRCUIT-SAT. Check that producing the circuit in the proof of the previous lemmas can be done in $O(T(n)\log T(n))$ time and $O(\log T(n))$ space. □

5.2 Function Complexity Classes, Counting Problems, and $\#\mathsf{P}$

We will now define some complexity classes of numerical functions on input strings rather than decision problems.

Definition 5.2. The class FP is the set of all functions $f : \{0,1\}^* \to \mathbb{N}$ that are computable in polynomial time.

Among the algorithmic problems representable by such functions are "counting problems" pertaining to common decision problems. For example,

Definition 5.3. Define #3-SAT as the problem of counting the number of satisfying assignments to a given 3-SAT formula ϕ.

Remark. Note that if we can solve #3-SAT in polynomial time, then clearly we can solve any NP-complete problem in polynomial time and thus $\mathsf{P} = \mathsf{NP}$.

We can define new complexity classes to represent such counting problems:

Definition 5.4. For any complexity class C, let $\#\mathsf{C}$ be the set of all functions $\{f : \{0,1\}^* \to \mathbb{N}$ for which there exists $R \in \mathsf{C}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $f(x) = \#\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\} = |\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\}|$.

In particular, for $\mathsf{C} = \mathsf{P}$, we obtain

Definition 5.5. Define the class $\#\mathsf{P}$ to be the set of all functions $\{f : \{0,1\}^* \to \mathbf{N}$ for which there exists $R \in \mathsf{P}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $f(x) = \#\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\} = |\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\}|$.

5.2.1 Function classes and oracles

Let us consider the use of oracle TMs in the context of function classes. For a language A, let FP^A be the set of functions $f : \{0,1\}^* \to \mathbb{N}$ that can be solved in polynomial time by an oracle TM $M^?$ that has A as an oracle.

Similarly, we can define oracle TM's $M^?$ that allow functions as oracles rather than sets. In this case, rather than receiving the answer from the oracle by entering one of two states, the machine can receive a binary encoded version of the oracle answer on an oracle answer tape. Thus for functions $f : \{0,1\}^* \to \mathbb{N}$ and a complexity class C for which it makes sense to define oracle versions, we can define C^f, and for a complexity class FC' of functions we can define $\mathsf{C}^{\mathsf{FC}'} = \bigcup_{f \in \mathsf{FC}'} \mathsf{C}^f$.

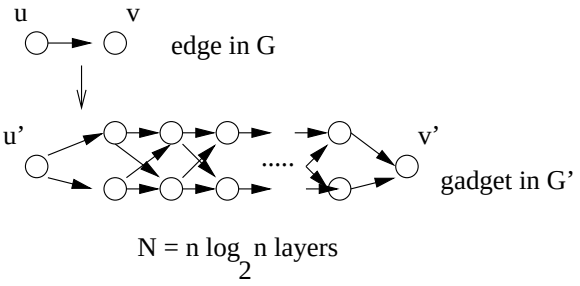

Figure 5.5: Gadgets used in reduction of #HAM-CYCLE to #CYCLE.

Definition 5.6. A function f is $\#\mathsf{P}$*-complete* iff

1. $f \in \#\mathsf{P}$.
2. For all $g \in \#\mathsf{P}, g \in \mathsf{FP}^f$.

As 3-SAT is NP-complete, #3-SAT is $\#\mathsf{P}$-complete:

Theorem 5.5. *#3-SAT is* $\#\mathsf{P}$*-complete.*

Proof. Cook's reduction is "parsimonious", in that it preserves the number of solutions. More precisely, in circuit form there is precisely one satisfying assignment for the circuit for each NP witness y. Moreover, the conversion of the circuit to 3-SAT enforces precisely one satisfying assignment for each of the extension variables associated with each gate. □

Since the standard reductions are frequently parsimonious, and can be used to prove $\#\mathsf{P}$-completeness of many counting problems relating to NP-complete problems. In some instances they are not parsimonious but can be made parsimonious. For example we have the following.

Theorem 5.6. #HAM-CYCLE *is* $\#\mathsf{P}$*-complete.*

The set of $\#\mathsf{P}$-complete problems is not restricted to the counting versions of NP-complete problems, however; interestingly, problems in P can have $\#\mathsf{P}$-complete counting problems as well. Consider #CYCLE, the problem of finding the number of directed simple cycles in a graph G. (The corresponding problem CYCLE is in P).

Theorem 5.7. *#CYCLE is* $\#\mathsf{P}$*-complete.*

Proof. We reduce from #HAM-CYCLE. We will map the input graph G for #HAM-CYCLE to a graph G' for #CYCLE. Say G has n vertices. G' will have a copy u' of each vertex $u \in G$, and for each edge $(u, v) \in G$ the gadget in Figure 5.5 will be added between u' and v' in G'. This gadget consists of $N = n\lceil \log_2 n \rceil + 1$ layers of pairs of vertices, connected to u and v and connected by $4N$ edges within. The number of paths from u' to v' in G' is $2^N > n^n$. Each simple cycle of length ℓ in G yields $2^{N\ell}$ simple cycles in G'. If G has k Hamiltonian cycles, there will be $k2^{Nn}$ corresponding simple cycles in G'. G has at most n^n simple cycles of length $\leq n-1$. The total number of simple cycles in G' corresponding to these is $\leq n^n 2^{N(n-1)} < 2^{Nn}$. Therefore we compute #HAM-CYCLE$(G) = \lfloor$#CYCLE$(G')/2^{Nn}\rceil$. □

The following theorem is left as an exercise:

Theorem 5.8. *#2-SAT is* $\#\mathsf{P}$*-complete.*

5.2.2 Determinant and Permanent

Some interesting problems in matrix algebra are represented in function complexity classes. Given an $n \times n$ matrix A, the determinant of A is

$$det(A) = \sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} \prod_{i=1}^{n} a_{i,\sigma(i)},$$

where S_n is the set of permutations of n and $sgn(\sigma)$ is the number of trnaspositions required to produce σ modulo 2. This problem is in FP.

The $sgn(\sigma)$ is apparently a complicating factor in the definition of $det(A)$, but if we remove it we will see that the problem actually becomes harder. Given an $n \times n$ matrix A, the *permanent* of A is equal to

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

Let 0-1PERM be the problem of finding the permanent of a 0-1 matrix. When we continue, we will prove the following theorem:

Theorem 5.9 (Valiant). *0-1PERM is #P-complete.*

The following is an interesting interpretation of the permanent. We can view the matrix A the adjacency matrix of a weighted bipartite graph on vertices $[n] \times [n]$ where $[n] = \{1, \ldots, n\}$. Each $\sigma \in S_n$ corresponds to a perfect matching of this graph. If we view the weight of a matching as the product of the weights of its edges the permanent is the total weight of all matchings in the graph.

In particular a 0-1 matrix A corresponds to an unweighted bipartite graph G for which A is the adjacency matrix, and Perm(A) represents the total weight of all perfect matchings on G. Let #BIPARTITE-MATCHING be the problem of counting all such matchings. Thus we obtain the following corollary as well:

Corollary 5.10. *#BIPARTITE-MATCHINGS is $\#$P-complete*