CMPT 710/407 - Complexity Theory
Lecture 4: Complexity Classes, Completeness, Linear Speedup, and Hierarchy Theorems

Valentine Kabanets

September 13, 2007

1 Complexity Classes

Unless explicitly stated, by a TM we will mean a *multi-tape* TM. When we talk about the space used by a multi-tape TM, we only count the number of cells on the worktape(s) that are touched by the TM during its computation. That is, we *do not* count the input tape or the output tape. We also assume that the input tape is *read-only*, and the output tape is *write-only* (with one-way access: after each write operation, the tape head moves one cell to the right, and can never move left).

Below n will denote the input size.

For some function $f : \mathbb{N} \to \mathbb{N}$,

- $\mathsf{Time}(f(n)) = \{L \mid L \text{ is decided by some TM in at most } f(n) \text{ steps}\}$
- $\mathsf{Space}(f(n)) = \{L \mid L \text{ is decided by some TM that touches at most } f(n) \text{ cells of its worktapes}\}$

For example,

- $\mathsf{P} = \cup_{k\geq 1}\mathsf{Time}(n^k)$,
- $\mathsf{EXP} = \cup_{k\geq 1}\mathsf{Time}(2^{n^k})$,
- $\mathsf{L} = \mathsf{Space}(\log n)$,
- $\mathsf{PSPACE} = \cup_{k\geq 1}\mathsf{Space}(n^k)$.

In general, a complexity class is a collection of languages decidable within a given amount of computational resources.

2 Completeness

For a complexity class C, we say that a language L is *C-complete* if

1. $L \in C$, and
2. every language in C is reducible to L.

Interpretation A C-complete language L is a "hardest" language in C (everybody else in C is at most as hard as L.)

Usefulness A C-complete language *captures* the complexity of the entire class C. So, we can reason about C by thinking about a single concrete problem from C.

We say that a language L is *C-hard* if every language in C is reducible to L (i.e., L may or may not be in C).

3 Linear Speedup Theorems

Theorem 1 (Linear Speedup: Time). *If a language L is decided by some TM in time $f(n)$, then, for any $\epsilon > 0$, there is a TM M' that decides L in time*

$$\epsilon f(n) + n + 2.$$

Proof Sketch. Encode k-tuples of symbols of M as *single* symbols of M', by increasing the alphabet of M'. Now, "compress" the input to M of length n to a new string (over the alphabet of M') of length n/k. This phase can be done in at most $n+n/k+2$ steps (copying the input in condensed form and returning the head of M' all the way to the left).

Now, the simulation phase begins: In just 6 steps of M', we can simulate k steps of the old machine M. First, in 4 steps (L,R,R,L), M' will gather and store in its finite state the symbols of M that can be scanned by M in at most k moves of M. With all that info, M' can compute what the tape of M would look like after k moves of M. Using at most two more steps (L,R or R,L), M' will change its tape appropriately.

The simulation phase needs at most $6\frac{f(n)}{k}$ steps. By setting $k = 6/\epsilon$, we get the total time of M' on input of size n is at most $\epsilon f(n) + n + 2$, as promised. □

Using similar ideas, one can also prove

Theorem 2 (Linear Speedup: Space). *If L is decided in space $f(n)$, then, for any $\epsilon > 0$, there is a TM deciding L in space $\epsilon f(n) + 2$.*

Exercise 3. *Give the proof of the Space Speedup Theorem.*

Conclusion Our model of computation is not refined enough to distinguish between time and space usage that differs by a constant. Hence, implicitly, we're always using the Big-Oh notation.

4 Time and Space

Motivating question: Given a Boolean formula with n nodes (and of height $\log n$), can we evaluate it in $O(\log n)$ space?

Try DFS? A naive implementation will use $O(\log^2 n)$ space. But, using the semantics of ANDs and ORs, we can do better. Note that if the left child of an OR gate is 1, then

we don't need to evaluate the right child. Similarly, for an AND gate, if its left child is 0, we don't need to evaluate its right child. Thus, for the algorithm that stops early (doesn't evaluate the right child) whenever possible, we can reconstruct the value of the left child without explicitly remembering it. All we need to remember now is our current position in the formula tree ($\log n$ bits) plus the value of the current node in the tree (another bit). This yields a $O(\log n)$ space algorithm. (Exercise: work out the details of this algorithm.)

What about a general Boolean circuit with n nodes? Can it be evaluated in space $O(\log n)$? We'll see that the answer is "Probably not" since otherwise every problem in P could be done in logarithmic space (since the Boolean circuit value problem is actually known to be P-complete under logspace reductions). The collapse of P to L seems extremely unlikely.

5 Hierarchy Theorems

First we recall a basic result from Computability Theory that there are undecidable languages. This can be argued as follows. There is a 1-1 correspondence between algorithms (TMs) and natural numbers. There is a 1-1 correspondence between languages (over the binary alphabet) and real numbers (in the interval $[0, 1]$). Since the set of reals is bigger than the set of natural numbers (as argued by Cantor a couple of hundred years ago), we have the existence of an undecidable language.

The problem with the argument above is its non-constructiveness. We haven't exhibited a particular example of a language that is not decidable. Turing was the first to give such examples. The most famous is his Halting Problem.

Recall $Halt = \{(M, x) \mid \text{TM } M \text{ accepts } x\}$. (This is a version of the Halting Problem. One could also define $Halt$ as the set of (M, x) such that M halts on x. This would also be undecidable.)

Theorem 4. *Halt is undecidable.*

We will prove the theorem in two stages. First, we define a certain language $Diag$ and show that $Diag$ is undecidable. Then we will show that $Diag$ is reducible to $Halt$, and hence, $Halt$ is also undecidable. (Note the use of a reduction to prove a lower bound on the complexity of $Halt$.)

To define $Diag$, we consider an enumeration $x_1, x_2, x_3, \ldots$ of all binary strings. Let $M_1, M_2, M_3, \ldots$ be an enumeration of all Turing machines whose descriptions are $x_1, x_2, x_3, \ldots$. (Note some binary strings correspond to "broken" TMs. We will assume that a broken TM does not accept any string.) We define $Diag$ as follows:

$$Diag = \{M \mid M \text{ does not accept input } M\}.$$

Lemma 5 (Turing). *Diag is undecidable.*

Proof. Our proof is by contradiction. Suppose some TM D decides $Diag$. Then we have one of two cases:

1. D accepts D, OR

2. D does not accept D.

In the first case, we get $D \notin Diag$ (by definition of $Diag$). Since D decides $Diag$ (by our assumption), D does not accept D. A contradiction.

In the second case, we get $D \in Diag$ (by definition of $Diag$). Since D decides $Diag$ (by our assumption), D accepts D. Again a contradiction.

Thus, our assumption that $Diag$ is decidable must be wrong. □

Proof of Theorem 4. Now we reduce $Diag$ to $Halt$. If $Halt$ were decidable by some TM H, then we could decide $Diag$ as follows:

> "On input M, run the TM H on (M, M), and negate the output (i.e., Accept if H rejects, and Reject if H accepts)."

It is easy to see that the described algorithm decides $Diag$. But, by our Lemma 5, $Diag$ is undecidable. Hence, so is $Halt$. □

The proof of Theorem 4 uses two very important techniques:

- simulation, and
- diagonalization.

The same ideas are used to prove *Hierarchy Theorems*: with more time (space), one can compute more languages.

In this course, we will always use *proper* complexity functions $f(n)$. A function $f(n)$ is called *proper* if there is a TM M that, on input 1^n, outputs exactly $f(n)$ symbols and runs in time $O(f(n) + n)$ and space $O(f(n))$. The usual functions like $\log n$, $\sqrt{n}$, n^2, 2^n, $n!$ are proper. Also, if f and g are proper, then so are $f + g$, fg, $f(g)$, f^g, 2^g.

Lemma 6. *Let $f(n)$ be any proper complexity function. The language*

$$Halt_f = \{(M, x) \mid \text{ TM } M \text{ accepts } x \text{ in at most } f(|x|) \text{ steps}\}$$

is decidable in time $g(n) = (f(n))^3$.

Proof. We can construct a universal TM H with 3 tapes that does the following. Given an input (M, x), our TM H converts a (possibly multi-tape) TM M to an equivalent one-tape TM M'. If M accepts x in at most $f(|x|)$ steps, then the new TM M' will accept x in at most $(f(|x|))^2$ steps.

Next, H will simulate M' on x for at most $(f(|x|))^2$ steps. Each step of M' can be simulated by H in at most $|M'|$ steps (i.e., the size of the description of the TM M', which is some constant dependent on M'); this constant is at most $f(|x|)$. Thus, our entire simulation will take at most $(f(|x|))^3$ steps. (Note that we needed $f(n)$ to be a proper function in order to be able to simulate M for at most $f(n)$ steps!) □

Consider the language

$$Diag_f = \{M \mid \text{ TM } M \text{ does not accept input } M \text{ in at most } f(|M|) \text{ steps}\}$$

By Lemma 6, the language $Diag_f$ is in $\mathsf{Time}(g(2n))$.

Theorem 7. *$Diag_f$ is not in* $\mathsf{Time}(f(n))$.

Proof. The proof is virtually the same as the one showing that the language *Diag* (defined earlier) is undecidable. The details are left as an exercise. □

Hence, we have

Theorem 8 (Time Hierarchy). *For every proper complexity function $f(n) \geq n$,*

$$\mathsf{Time}(f(n)) \subsetneq \mathsf{Time}((f(2n))^3).$$

Similarly, we can prove

Theorem 9 (Space Hierarchy). *For every proper complexity function $f(n) \geq \log n$,*

$$\mathsf{Space}(f(n)) \subsetneq \mathsf{Space}((f(n)) \log f(n)).$$

As a simple application of these Hierarchy Theorems, we can prove that $\mathsf{P} \subsetneq \mathsf{EXP}$ and $\mathsf{L} \subsetneq \mathsf{PSPACE}$. (Do you see why?)